CAMPUS
DE EXCELENCIA
INTERNACIONAL

Graduado en Ingeniería Informática

Universidad Politécnica de Madrid

ETSIINF

TRABAJO FIN DE GRADO

Desarrollo de una interfaz para controlar por voz la aplicación móvil de mensajería *Telegram*

AUTOR: Lydia Galán Pache

TUTOR: Ángel Herranz Nieva

Madrid, Junio 2014

Resumen

La aparición de los *smartphones*, trajo consigo el desarrollo de aplicaciones móviles de mensajería instantánea. Estas aplicaciones aprovechan la infraestructura de las redes de datos para enviar los mensajes de unos dispositivos a otros, lo que supone la posibilidad de enviar mensajes ilimitados a bajo coste. Hoy en día lo inusual es ver a alguna persona que haga uso de los antiguos mensajes de texto o *sms* (Short Message Service), que además llevan el coste de comunicación definido por las distintas operadoras. Tanto ha sido su auge que se ha convertido en uno de los principales medios de comunicación tanto en el ámbito personal como empresarial. Desafortunadamente, cada vez son más los conductores que hacen uso de las aplicaciones de mensajería para enviar y recibir mensajes mientras conducen, a pesar de que su uso está totalmente prohibido y penado por la ley.

Por este motivo, en este proyecto se propone la modificación de la aplicación de mensajería *Telegram*, que permite controlar el envío y recepción de mensajes únicamente utilizando la voz, evitando así cualquier tipo de distracción ocasionada por la interacción táctil con el dispositivo.

Esta idea propuesta en el proyecto puede ayudar a reducir el número de accidentes ocasionados por este tipo de distracciones al volante, así como las posibles multas e incidentes que pueda ocasionar el uso del móvil durante la conducción.

Abstract

The emergence of *smartphones*, fostered the development of mobile instant messaging applications. These applications take advantage of the infrastructure of data networks to send messages between devices with almost no additional cost attached to it. Today you will hardly be able to find a person who makes use of the old text messages or *sms* (Short Message Service), and therefore bears the cost of communication defined by the respective operators. This boom has been such that it has become one of the main communication methods or channels in both the personal and work environments. Unfortunately, more and more drivers use messaging applications to send and receive messages while they are driving, even though its use is strictly prohibited and punished by law.

Therefore our objective is to modify the existing messaging application *Telegram* allowing interaction with the mobile device by only using the user's voice to send and receive messages, avoiding any distractions that any tactile interaction with the device could cause.

The aim is to significantly try to reduce accidents caused by this type of distractions while driving, as well as to avoid any related potential fines and incidents that may result from use of mobile phone while driving.

Índice general

Índice de figuras

Capítulo 1

Introducción

En el presente documento se detalla el proceso seguido en el desarrollo de un módulo para controlar por voz una de las aplicaciones de mensajerías ya existentes hoy en día, concretamente, *Telegram* [19]. El proceso comenzará con la puesta en conocimiento de la plataforma Android en general, así como de las herramientas que esta posee para el reconocimiento y síntesis de voz dentro de las aplicaciones móviles. Posteriormente, se desarrollará el módulo que permitirá interactuar con la aplicación *Telegram* utilizando únicamente la voz.

1.1. Motivación

Con el paso del tiempo los hábitos de las personas han ido cambiando. Hace algunos años era común el uso de mensajes de texto (*sms*) para comunicarse de una manera fácil, rápida y económica. Actualmente, existen aplicaciones de mensajería instantánea que aprovechan las infraestructuras de las redes de datos para permitir los envíos de mensajes sin coste adicional a la tarifa de datos del usuario.

Whatsapp [20], *Telegram*, *LINE* [18] son claros ejemplos de aplicaciones que cuentan con miles o millones de usuarios. Concretamente, según muestran los datos analizados por Flurry Analytics [13] en la Figura 1.1, durante el pasado año 2013 se produjo un incremento del 115 % en la utilización de las aplicaciones móviles para comunicarse. Ese incremento en el número de usuarios ha generado que el teléfono móvil sea utilizado en actividades como por ejemplo la conducción, lo cual está prohibido y penado por la ley.

El último estudio realizado por CDC (Centers for Disease Control and Prevention) [5, 6] en el año 2011 compara el porcentaje de conductores distraídos en Estados Unidos

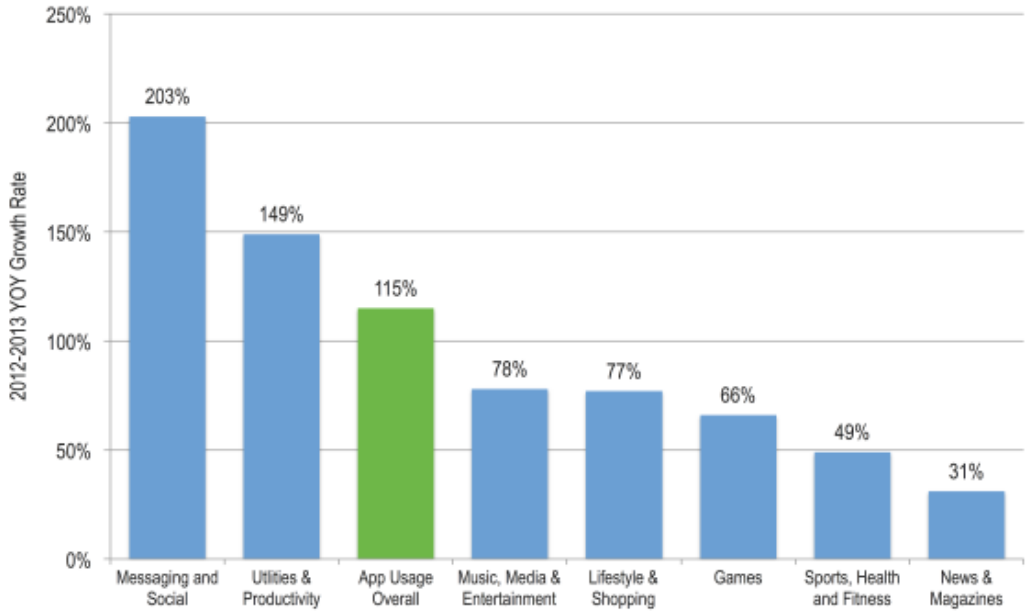

Figura 1.1: Estadística de uso del móvil por categorías [13].

y siete países europeos (Bélgica, Francia, Alemania, Holanda, Portugal, España y el Reino Unido). La encuesta fue completada por 10.338 ciudadanos europeos y 11.806 ciudadanos estadounidenses, y los resultados muestran que conductores entre 18-64 años afirman que: un 21 % en el Reino Unido y un 69 % en Estados Unidos han hablado por teléfono durante la conducción; un 15 % en España, 31 % en Portugal y 31,2 % en Estados Unidos habían leído y escrito emails y mensajes de texto en los últimos 30 días antes de realizar las encuestas. Estos datos se muestran en la Figura 1.2 y Figura 1.3.

Haciendo más hincapié en datos relativos a nuestro país publicados por el periódico *ABC* [9] en ese mismo año 2011, la distracción con un 39 % del total, aparece como primer factor en 32.497 accidentes con víctimas, en los cuales fallecieron 904 personas y 4.590 resultaron heridas graves. Debido a estos preocupantes datos la Dirección General de Tráfico realizó una campaña de "concienciación de los peligros que suponen las distracciones al volante" en la semana del 3 al 9 de junio de 2013. Según confirmó el periódico *El País* [12], en dicha semana, se controlaron 357.000 vehículos y se denunció a un total de 4.312 conductores. El 78 % de los automovilistas denunciados lo fueron por usar el teléfono móvil mientras conducían.

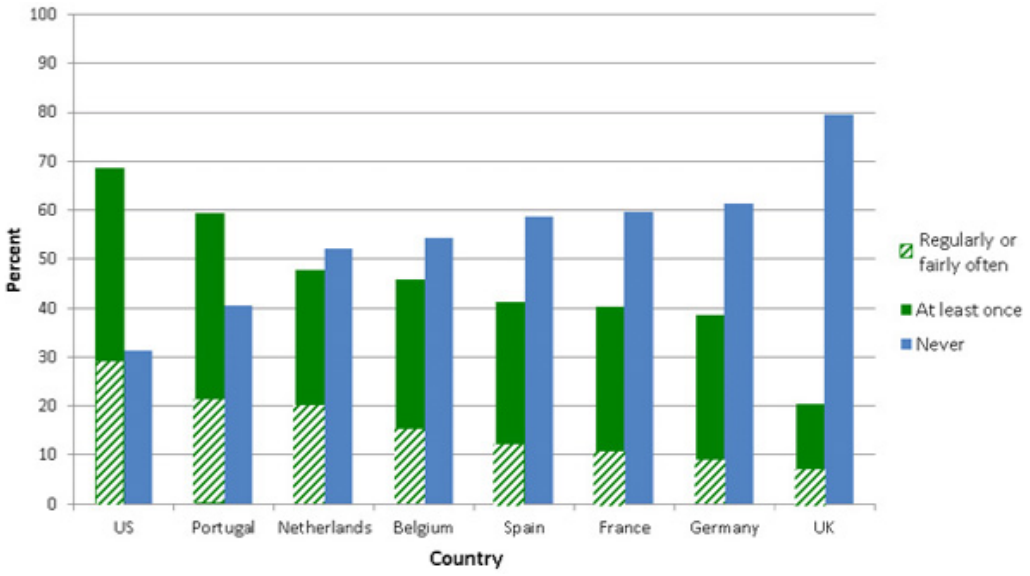

Figura 1.2: Porcentaje de personas que hablaron por el móvil estando al volante [5].

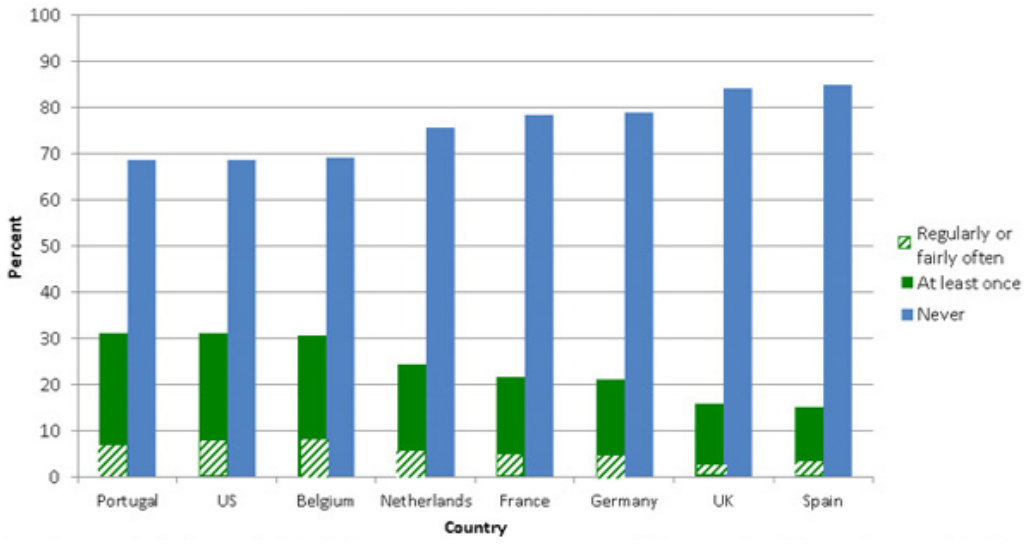

Figura 1.3: Porcentaje de personas que escribieron un mensaje estando al volante [5].

Observando estos datos y estadísticas, y teniendo en cuenta su tendencia ascendente, podemos concluir que aún con las prohibiciones existentes en el uso de teléfonos móviles durante la conducción este porcentaje no se va a ver reducido. Por esta razón, la motivación de este proyecto es la de mejora de las actuales aplicaciones móviles de mensajería para poder utilizarlas reduciendo los riesgos de accidente.

1.2. Objetivos del Proyecto

El proyecto se centra en la modificación del código fuente de *Telegram* con el fin de añadir la funcionalidad para controlar por voz el cliente de mensajería y evitar la interacción táctil cuando se reciba un mensaje, y se desee contestar de forma inmediata. Para alcanzar con éxito el propósito del proyecto se han definido los siguientes objetivos parciales:

- Analizar el API de Android para el reconocimiento y síntesis de voz en aplicaciones móviles.
- Analizar la aplicación de mensajería *open-source Telegram* y determinar las modificaciones necesarias para interaccionar por voz.
- Desarrollar un módulo que permita leer con voz sintética los mensajes recibidos.
- Completar el módulo para poder interactuar con *Telegram* por comandos de voz y convertir a texto mensajes que el usuario haya dictado.

1.3. Estructura del documento

Como se ha podido observar, en primer lugar, se ha puesto en situación al lector de cuál es la motivación del proyecto y en qué consiste la realización del mismo.

En el capítulo 2 se hace una revisión de los distintos clientes de mensajería existentes así como de las distintas aplicaciones capaces de leer otros tipos de mensajes como emails o mensajes de texto.

En el capítulo 3 describimos la aplicación *Telegram* en el contexto en el que va a ser modificada. Además, se estudian y describen las distintas librerías o APIs utilizadas para poder desarrollar el módulo de interacción por voz.

Posteriormente, en el capítulo 4 se detalla de manera específica todas las modificaciones e implementaciones realizadas al código que existe actualmente sobre esta aplicación de open-source software.

Finalmente, en el capítulo 5 se exponen los resultados obtenidos sobre las pruebas realizadas al módulo implementado, así como las conclusiones finales de la elaboración de todo el proyecto y el posible trabajo futuro en el capítulo 6.

Capítulo 2

Estado del Arte

2.1. Aplicaciones de mensajería instantánea

Durante los últimos cinco años hemos visto el creciente auge del uso del móvil que continúa aumentando año tras año. Como ya hemos comentado anteriormente, según estudios realizados por Flurry Analytics [13] el uso de las aplicaciones móviles se incrementó en un 115 % el pasado 2013. Concretamente, según las estadísticas, el uso de aplicaciones de mensajería se incrementó en un 203 %, y lo que es más sorprendente, su uso se triplica año tras año. En noviembre de 2013, Benedict Evans publicó su estudio llamado "The mobile is eating the world" [4]. En él mostró estadísticas como el uso de aplicaciones móviles de redes sociales y mensajería que se muestran en la Figura 2.1.

A continuación, la Figura 2.2 muestra una comparativa, publicada por el periódico *ABC* [10], con las especificaciones de las aplicaciones móviles más utilizadas hoy en día.

Cabe mencionar, además, algunas características y datos históricos de algunas estas aplicaciones de mensajería:

- **Whatsapp**
  - Creada por los americanos Brian Acton y Jan Koum.
  - Actualmente la aplicación de mensajería instantánea más famosa que existe.
  - Lleva en funcionamiento desde el año 2009.
  - En agosto de 2012, procesaba una media de 10 billones de mensajes al día. En junio de 2013 rompió su record procesando 27 billones de mensajes al día.
  - En diciembre de 2013 Whatsapp proclamó la posesión de 400 millones de usuarios activos.

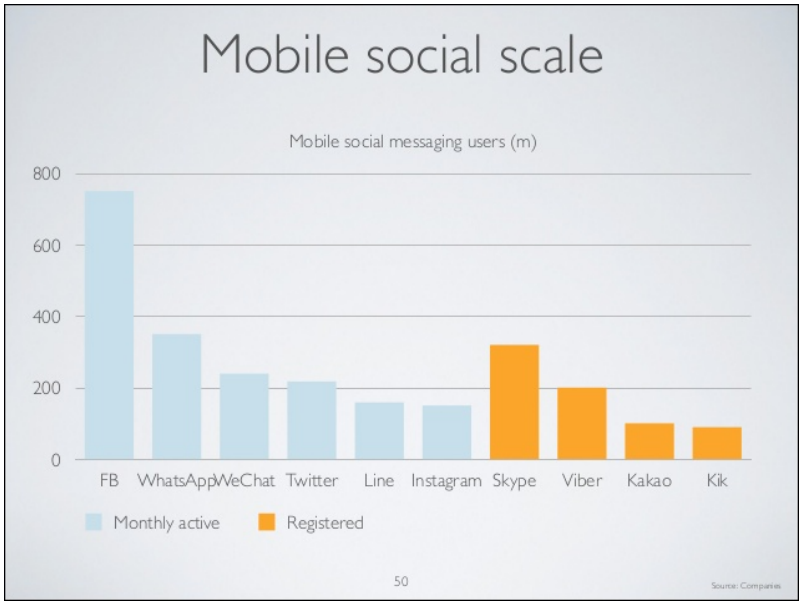

Figura 2.1: Usuarios activos o registrados en alguna aplicación de mensajería [4].

- Multiplataforma. El software está disponible para Android, Blackberry OS, Apple iOS, Nokia, Symbian, Windows Phone, Blackberry10.

- **LINE**
  - Creada por "LINE Corporation", compañía sur coreana-japonesa.
  - Lanzada al mercado en 2011.
  - Dieciocho meses tras su aparición alcanzó los 100 millones de usuarios registrados. En noviembre de 2013 anunció 300 millones de registros a nivel mundial, 50 millones de los cuales eran japoneses.
  - Multiplataforma. El software está disponible para Android, Blackberry OS, Apple iOS, Nokia, Firefox OS, Windows Phone, MS Windows, Mac OS X.
  - A diferencia de Whatsapp, posee versión de escritorio.

- **Telegram**
  - Creada por los hermanos rusos Nikolai y Pavel Durov.
  - La única de las aplicaciones de mensajería que es *open-source.*
  - Fundada en el año 2013.

- En octubre de 2013 contaba con 100.000 usuarios activos al día.
- En marzo de 2014, Telegram anunció 35 millones de usuarios registrados al mes y 15 millones de usuarios activos al día.
- Multiplataforma. El software está disponible para Android, Apple iOS, Windows Phone, MS Windows, Linux, Mac OS X.
- Se trata de la aplicación de mensajería más segura hasta el momento.

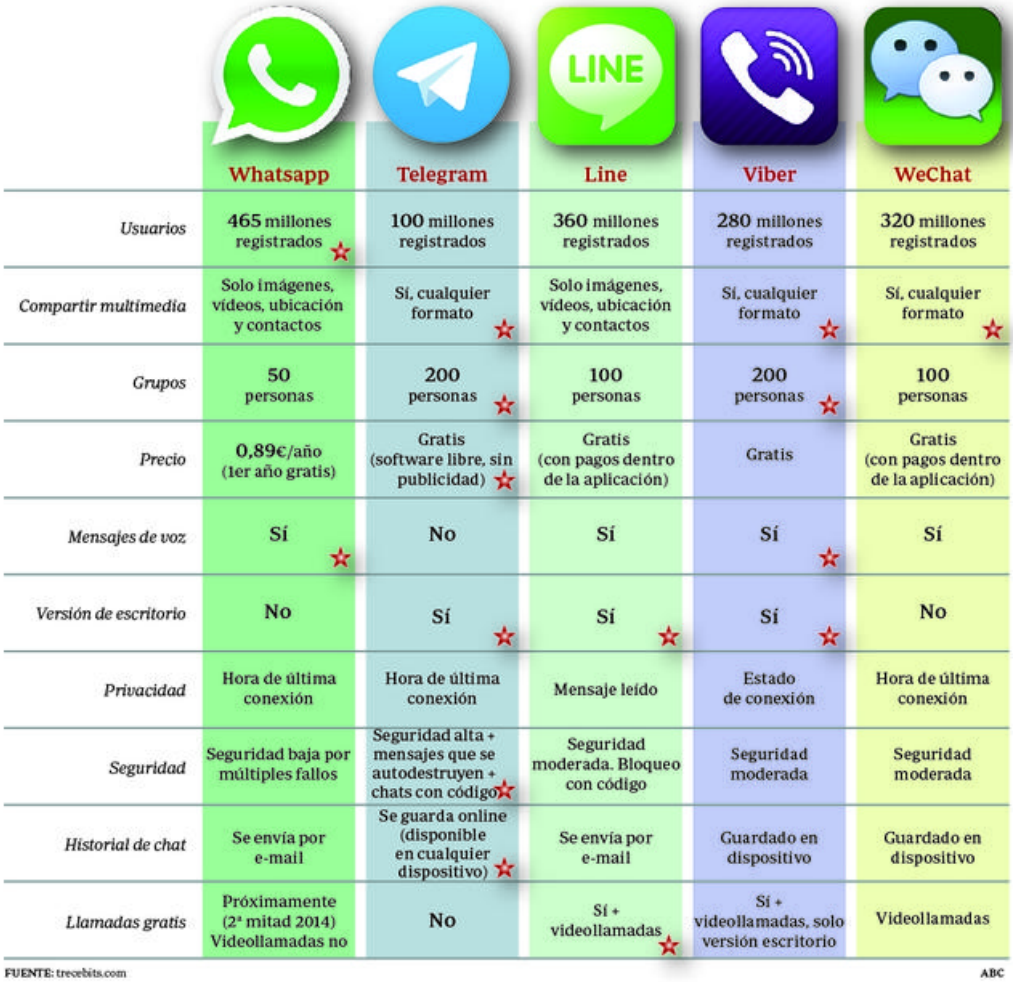

Figura 2.2: Características de las aplicaciones móviles de mensajería más usadas [10].

De todos los clientes de mensajería que se han revisado, se decidió desarrollar la interfaz de control por voz para la aplicación *Telegram*. El principal motivo para esta elección ha sido que tanto el cliente como el API de los servidores son *open-source*, por lo que se puede tener acceso al código fuente y a las llamadas remotas

para acceder a toda la funcionalidad proporcionada por *Telegram*. Además, este cliente presenta ciertas características que lo distinguen de sus competidores como la rapidez, la utilización de una infraestructura descentralizada, y una selección eficiente del servidor de comunicaciones basado en proximidad. Finalmente, otro punto a destacar de *Telegram* es que ofrece seguridad a través de canales de comunicación encriptados.

2.2. Aplicaciones que utilizan interfaz de voz

Hemos visto en el capítulo de introducción los datos que reflejan la reticencia que tienen muchos usuarios de apagar el móvil cuando se encuentran conduciendo. A raíz de esto, ya existen en el mercado algunas aplicaciones que intentan evitar toda distracción al volante que implique coger el móvil con el objeto de leer o contestar a un email o mensaje recibido. A continuación, se describen algunas de las aplicaciones que se utilizan en estas circunstancias.

Hatomico, acrónimo de **ha**go **to**do **mi**entras **co**nduces. El móvil detecta la velocidad de conducción y toma el control de los mensajes y llamadas entrantes. Avisa cuando llegan mensajes de correo o de mensajería instantánea y es capaz de reproducirlo. También permite descolgar el teléfono por voz o rechazar la llamada. Es configurable para decir a qué personas se atiende y a cuáles no. Para versiones de Android 4.4.2 y superiores también lee los mensajes de algunas aplicaciones de mensajería, como Whatsapp, Telegram y Hangouts, pero no permite contestar a ellos utilizando la voz. Es gratuita y sólo está disponible para dispositivos Android y no para todas las versiones de este último.

Text'nDrive, lee los mensajes de texto y correo electrónico en tiempo real. Se activa cuando el móvil detecta la velocidad. Existe además una versión *pro* de pago que permite contestar a los mensajes por voz. Disponible para dispositivos iPhone y Android.

Text STAR, aplicación que toma el control del teléfono. Retiene las llamadas y mensajes entrantes hasta que detecta que el automóvil se ha detenido por un tiempo prolongado. Permite configurar un mensaje de aviso a los contactos que se intenten comunicar de que se está al volante, para posponer sus llamadas o mensajes. Sólo disponible para dispositivos Android y sólo posee el idioma de inglés.

DriveSafe.ly, lee los mensajes entrantes y *sms* pero no permite responder por voz. Señala la procedencia de la llamada para que no se tenga que mirar la pantalla. Sólo disponible para Android.

Safe Driver, detecta de forma automática que se está conduciendo y suspende las

comunicaciones del móvil. De este modo evita que se reciban llamadas o *sms*, o mensajes de aplicaciones de mensajería. Además, se puede programar una contestación automática avisando de que se está al volante y se responderá tan pronto como se detenga. Aplicación exclusiva de Android.

STOPTxting, bloquea en el dispositivo móvil la aplicación de correo electrónico Gmail, el navegador y cualquier aplicación de mensajería cuando detecta cierta velocidad a través del GPS. De esta forma previene distracciones al volante.

Como hemos visto anteriormente, existen aplicaciones capaces de detener cualquier actividad que implique recibir algún tipo de mensaje o llamada durante la conducción, así como aplicaciones que crean auto-respuestas. Además, existen algunas aplicaciones como *Hatómico* o *Text'nDrive* que son capaces de leer los mensajes de texto y los correos electrónicos entrantes, y sin embargo, no permiten ningún tipo de interacción que permita contestar en tiempo real. Si se deseara contestar a estos mensajes mientras se está conduciendo habría que coger el teléfono móvil para tener que escribirlo, por lo que seguiría existiendo esa falta de seguridad al volante. Sólo la aplicación *Hatómico* es capaz de pronunciar mensajes de otras aplicaciones de mensajería como Whatsapp o Telegram, pero ninguna aplicación por el momento es capaz de contestar a los mensajes recibidos a través de interfaces de voz.

Con la realización de este proyecto se intentaría suplir la falta o posibilidad de interacción que hay con las aplicaciones existentes actualmente, de manera que se pueda contestar a los mensajes recibidos sin tener que interactuar de forma táctil con el dispositivo, lo que supone una distracción para el conductor.

Capítulo 3

Análisis

Antes del desarrollo del módulo de Control por Voz se ha hecho una fase de análisis tanto de la aplicación cliente de *Telegram* para la plataforma de Android, como de las librerías o APIs que Android proporciona para su desarrollo.

3.1. Análisis de Requisitos

En primer lugar, realizamos un análisis de requisitos para definir las funcionalidades que debe realizar el módulo.

CARACTERÍSTICAS DE LOS USUARIOS

El sistema se creará para todo tipo de usuarios que deseen interactuar por voz con la aplicación de mensajería *Telegram*. Indicado sobre todo para aquellos usuarios que deseen utilizar el dispositivo móvil mientras se encuentren conduciendo, sin tener que interaccionar de forma táctil con él.

SUPOSICIONES Y DEPENDENCIAS

Los usuarios deberán disponer de un dispositivo móvil con Sistema Operativo Android. Necesitarán conexión a Internet, ya sea vía *Wi-Fi* o a través de tarifa de datos.

REQUISITOS DE FUNCIONALIDAD

R01. El módulo se realizará para la aplicación de *Telegram* en la plataforma de Android.

R02. El sistema permitirá sintetizar con voz los mensajes recibidos por la aplicación.

R03. El sistema reproducirá el nombre del usuario emisor del mensaje.

R04. El sistema informará al usuario de las posibles comandos por voz en caso de que no se conozcan.

R05. El sistema escribirá y dejará constancia en el chat de los mensajes que se envíen por voz.

R06. La aplicación debe permitir activar y desactivar la funcionalidad de Control por Voz.

R07. El sistema funcionará y notificará los mensajes incluso con la aplicación de *Telegram* cerrada o en segundo plano.

R08. Se debe permitir al usuario interactuar por distintos comandos de voz:

> **R08.1**. Comando “responder”: El usuario podrá responder al mensaje que se ha recibido en ese momento.
>
> **R08.2**. Comando “repetir”: El sistema repetirá el mensaje que se acabar de recibir.
>
> **R08.3**. Comando “enviar”: El sistema permitirá enviar un mensaje a cualquier contacto que también posea la aplicación de *Telegram*. La selección del contacto también debe realizarse por voz.

RESTRICCIONES DE DISEÑO

R09. El módulo se realizará modificando el código fuente de *Telegram* intentando mejorar la cohesión funcional y minimizar el acoplamiento.

ATRIBUTOS DEL SISTEMA

R10. Dispositivos móviles que tengan Sistema Operativo Android.

3.2. Descripción de Telegram

Como se mencionó en capítulos anteriores, *Telegram* posee unas características que cabe destacar y que describiremos a continuación brevemente [14, 16].

3.2.1. Características principales de *Telegram*

Open-source

Como ya adelantamos en el capítulo 2 de "Estado del Arte", la principal característica de la aplicación *Telegram* es el hecho de ser *open-source*, tanto en el protocolo de comunicación con el servidor, como en el acceso al código fuente de esta aplicación multiplataforma desde la página web de *Telegram* [16]. En este proyecto nos hemos centrado en la parte del código fuente, quedando modificada por tanto, la aplicación cliente para móvil.

Analizamos las clases de código intentando llegar a la conclusión de cuáles son las modificaciones necesarias a realizar para implementar el control por voz en el envío y recepción de mensajes. Concretamente, existen una serie de clases de este código fuente que resultan especialmente relevantes para implementar la funcionalidad del proyecto, y que describiremos más adelante.

Seguridad y Encriptación

Telegram es más segura que otras aplicaciones de mensajería que existen actualmente en el mercado. Su seguridad se basa en el protocolo *MTProtocol* [15], inventado por Nikolai Durov, uno de los creadores de la aplicación. La base de este protocolo es el empleo de algoritmos que hacen compatibles una alta seguridad con rapidez y fiabilidad.

Además, *Telegram* permite crear chats encriptados utilizando cifrado extremo a extremo, lo que significa que sólo la persona que envía y la persona que recibe los mensajes podrá leer dichos mensajes, sin dejar rastro de ellos en los servidores y por tanto, sin quedar almacenados. Estos chats encriptados permiten utilizar mensajes de autodestrucción (desaparecen automáticamente de la imagen del chat tras un tiempo definido por el usuario) y no permiten su reenvío. Dan soporte a dos tipos de encriptación segura: servidor-cliente/cliente-servidor para el caso de los chats ordinarios, y cliente-cliente en el caso de que los chats sean secretos. Su cifrado se basa en la utilización del algoritmo de encriptación AES simétrico de 256 bits, el algoritmo RSA 2048 y el

algoritmo Diffie-Hellman que está basado en el intercambio de claves.

Todo esto permite un envío y recepción segura de mensajes, archivos, localizaciones, vídeos y fotos de cualquier formato a través de la aplicación.

Tecnología en la *nube*

Telegram posee un sistema *cloud*, basado en la nube, que almacena contactos, mensajes, archivos, fotos, de tal forma que el usuario puede acceder a ellos desde distintos dispositivos (incluido el ordenador de sobremesa). En el caso de desinstalar la aplicación o cambiar de teléfono no se perderían datos ni conversaciones.

Infraestructura Descentralizada

Otra característica destacable de *Telegram* es su rapidez. Esto se consigue gracias a que utiliza los centros de datos ubicados en todo el mundo de manera que un dispositivo se conectará con el servidor más cercano para enviar el mensaje, siendo un proceso más eficiente.

En las próximas secciones hablaremos sobre las APIs necesarias para llevar a cabo el propósito del proyecto.

3.2.2. Clases relevantes del código fuente de *Telegram*

Con el objetivo de realizar la implementación del módulo de Control por Voz, primero realizamos un análisis del cliente de la aplicación de *Telegram* para la versión de Android después de su descarga de la página de *GitHub* [17], proporcionada por los creadores de la aplicación.

- **org.Telegram.ui:ApplicationLoader.java**

  Clase que inicializa la aplicación, es la actividad principal. Se implementa el método *onCreate()*, invocado cuando la aplicación comienza, antes de cualquier servicio o actividad. Esta clase tiene mucha importancia pues permite registrarse en el sistema operativo para recibir notificaciones como puede ser la recepción de un mensaje en la aplicación de *Telegram* con el móvil apagado. Al encender el dispositivo aparece la notificación de mensaje recibido aunque la aplicación no haya sido arrancada manualmente por el usuario. En ella se ejecutan todas las configuraciones que *Telegram* posee por defecto.

El control por voz debe funcionar incluso cuando la aplicación esté pausada o apagada, por ello la inicialización se deberá realizar en esta clase.

- **org.Telegram.messenger:NotificationCenter.java**

  Telegram en su mayoría funciona a través de eventos. La clase NotificationCenter implementa los métodos *addObserver()* y *removeObserver()*, necesarios para añadir y eliminar observadores o suscriptores a varios tipos de eventos. Además, esta clase define la interfaz *NotificationCenterDelegate*, la cual actúa como delegado encargándose de notificar, a varios componentes de la aplicación, de un determinado evento con el fin de que dichos componentes tomen alguna acción al respecto. Define el método *didReceivedNotification()*, método que debe ser implementado por todas las clases que deseen realizar una acción específica al recibir un evento. Con eventos nos referimos a acciones como recibir un mensaje nuevo, crear un chat para hablar con un contacto o recibir un mensaje del Servidor, entre otros.

- **org.Telegram.messenger:MessagesController.java**

  Clase donde se implementa la mayor parte de la funcionalidad sobre los mensajes que se envían. En ella se definen e inicializan la mayor parte de las estructuras de datos que almacenan los distintos *chats*, *users*, *dialogs*, *encryptedChats*, entre otras. Dichas estructuras se encargan de mantener el estado con el servidor a la hora de realizar las comunicaciones.

  De igual forma, se definen las constantes que actúan como identificadores de notificaciones de eventos, como por ejemplo *didReceivedNewMessages*, *updateInterfaces*, *messagesReaded*, *messageReceivedByAck*, etc.

  Además, se implementan numerosos métodos, como por ejemplo el enviar distintos tipos de mensaje (archivo, imagen, video, localización), comenzar un chat, añadir un usuario a un chat, borrar un usuario de un chat, almacenar los mensajes en la nube, marcar un mensaje como leído, así como todo lo relacionado con los chats secretos.

- **org.Telegram.messenger:ContactsController.java**

  Clase que gestiona la sincronización con *Telegram* de los contactos guardados en el dispositivo móvil. En ella se definen todas las estructuras de datos encargadas de almacenar los contactos de *Telegram*, como pueden ser *contacts*, *contactsBook*, *contactsBookSPhones*, *sortedContactsSectionsArray* o *contactsByPhone*.

  Es muy importante conocer qué datos se almacenan del contacto a la hora de enviar un mensaje. Esta clase nos será útil para poder acceder a los contactos del usuario.

- **org.Telegram.messenger:ConnectionsManager.java**

  Clase que configura todas las conexiones con el servidor. En ella se definen las estructuras encargadas de gestionar las conexiones, ejemplos de estas estructuras son: *datacenters*, *sessionsToDestroy*, *processedSessionChanges*, *messagesIdsForConfirmation* o *quickAckIdToRequestIds*. Además, esta clase define los métodos necesarios para guardar, comenzar o destruir la conexión: *loadSession()*, *saveSession()*, *getNewSessionId()*, entre otros.

- **org.Telegram.messenger:TLRPC.java**

  Define los objetos Java de la información que se intercambia entre el servidor y el cliente. En ella se implementan todas las clases de los objetos que se pueden enviar y recibir por parte del servidor: mensajes ordinarios, contactos, mensajes encriptados. Si se produce algún error en las comunicaciones también es esta clase la que se encarga de notificarlos y gestionarlos. Ejemplos de clases implementadas en TLRPC.java son *BadMsgNotification*, *TL_bad_server_salt*, *TL_error*, *TL_msgs_ack* o *messages_SentEncryptedMessage* entre otras muchas.

- **org.Telegram.ui:ChatActivity.java**

  Clase que se encarga de gestionar todo lo relacionado con la creación, progreso, actualización y cierre de un chat en *Telegram*. Es decir, es la clase que se encarga de mostrar el mensaje o los mensajes cuando se abre el chat de una conversación mantenida con otra persona. Actualiza la interfaz de la pantalla del chat. Además, se encarga de almacenar el estado de la aplicación para que una vez eliminado el proceso de la aplicación, se pueda recuperar su estado cuando el usuario vuelva a ella. Es decir, permite que no se pierda toda la información de los chats que el usuario mantiene en *Telegram*.

  Al gestionar toda la visualización de los mensajes, esta clase nos va a ser muy útil para el desarrollo del Control por Voz.

- **org.Telegram.ui:SettingsActivity.java**

  Clase que implementa la parte de "Ajustes" de *Telegram*. En ella se crea toda su interfaz, iconos y elementos interactivos necesarios para poder configurar personalmente la aplicación en cada dispositivo móvil. La figura 3.1 muestra la interfaz que posee el apartado "Ajustes" de *Telegram*.

  Esta clase se deberá modificar para poder añadir una nueva funcionalidad de "Control por Voz". De esta manera el usuario podrá configurar esta opción siempre y cuando lo desee.

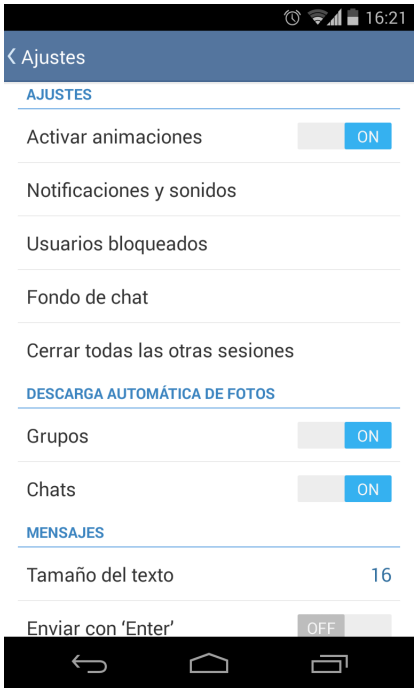

Figura 3.1: Interfaz de "Ajustes" de *Telegram.*

3.3. Descripción Android TextToSpeech

Para llevar a cabo la realización de la síntesis de voz se ha utilizado el API de Android *android.speech.tts* [3]. Esta API contiene una serie de clases, métodos e interfaces capaces de realizar la síntesis de voz de un texto. A continuación, vamos a describir las clases así como las interfaces más significativas para la implementación de este proyecto.

3.3.1. Clases e Interfaces de *android.speech.tts*

TextToSpeech, clase pública que posee los principales métodos para llevar a cabo la síntesis de voz de un texto determinado. En concreto se utiliza el método *speak()*. Posee también métodos necesarios para acceder a las características de cada dispositivo que utilice sistemas operativos Android. Así, métodos como *getLanguage()* nos permite conocer el idioma que está siendo utilizado en el teléfono móvil para poder reproducir el texto en el idioma adecuado. De igual forma *setLanguage()* nos permite modificar directamente el idioma en el que se desee realizar la síntesis de voz.

TextToSpeech.Engine, clase que contiene los parámetros y constantes necesarias para llevar a cabo un seguimiento y control del text-to-speech. Lo primero que se debe

realizar en la implementación es la inicialización y activación del motor de *TextToSpeech*. La constante *ACTION_CHECK_TTS_DATA* verifica la correcta activación del motor de síntesis de voz en el dispositivo. En ocasiones puede que muchos de los archivos necesarios para su correcto funcionamiento no estén instalados. En ese caso, la constante *ACTION_INSTALL_TTS_DATA* lanza automáticamente la actividad correspondiente para descargar del *Play Store* e instalar lo que sea requerido.

El parámetro *EXTRA_AVAILABLE_VOICES* permite obtener un *ArrayList* de todos los idiomas disponibles en el teléfono móvil y que pueden ser utilizados.

TextToSpeech.OnInitListener, interfaz que define el método de *callback onInit()*. Con la implementación de este método nos aseguramos de que el motor de TextToSpeech se haya inicializado con éxito.

UtteranceProgressListener, clase abstracta que actúa de *callback* sobre los eventos relacionados con el progreso de la síntesis de voz. El método *onDone()*, por ejemplo, es llamado cuando el texto ha sido pronunciado y la síntesis de voz ha finalizado con éxito. El método *onError()* es llamado si ocurre algún error durante el proceso de pronunciación.

En la figura 3.2 se muestra el diagrama que clarifica lo explicado anteriormente. Cabe destacar que no es el diagrama UML de toda la API de Android *android.speech.tts*, sino sólo la parte más relevante para este proyecto.

3.4. Descripción Android SpeechRecognizer

El reconocimiento de voz se lleva a cabo gracias a la API de Android llamada *android.speech* [2]. Al igual que en el caso de la librería *android.speech.tts*, ésta posee las clases, métodos e interfaces necesarias para llevar a cabo la implementación de la parte del proyecto que implique el reconocimiento de la voz del usuario. A continuación, describiremos las clases más relevantes.

3.4.1. Clases e Interfaces de *android.speech*

SpeechRecognizer, clase pública que contiene los métodos necesarios para realizar la inicialización y llevar a cabo un seguimiento del progreso del servicio de reconocimiento de voz. Concretamente, la inicialización se lleva a cabo con el método *createSpeechRecognizer()*. Una vez se haya creado e iniciado con éxito, podremos utilizar el método *startListening()* que nos permitirá comenzar a escuchar a la persona que interaccione con el dispositivo móvil. Cabe destacar que esta API transmitirá el audio a servidores remo-

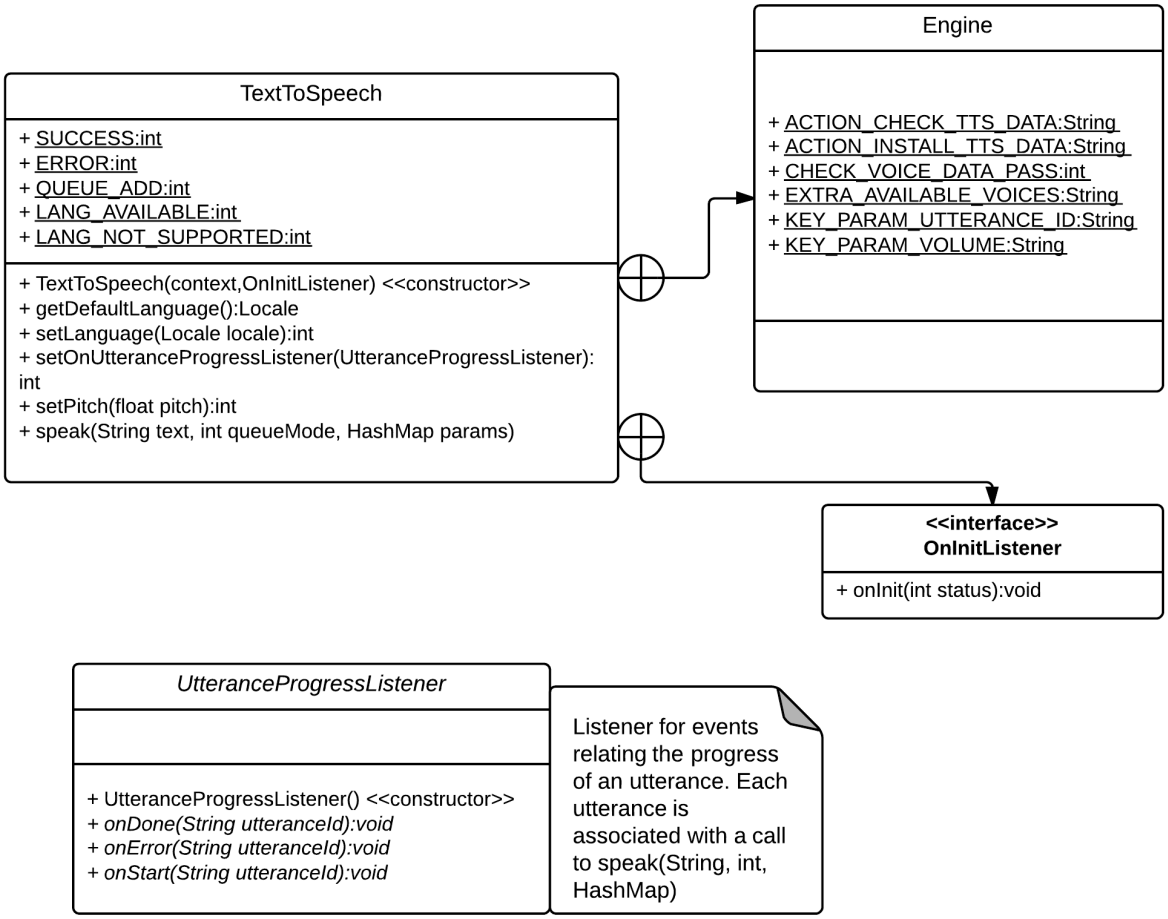

Figura 3.2: Diagrama UML *TextToSpeech*.

tos de Google para realizar el reconocimiento de voz. Del mismo modo, serán necesarios eventos de *callback* para seguir el progreso del *SpeechRecognizer*. Con el método *setRecognitionListener()* definiremos qué clase recibirá los eventos de *callback*.

Además, esta clase contiene las constantes que identifican los errores que pueden aparecer en el progreso del reconocimiento de la voz. Ejemplos de estás constantes son, *ERROR_AUDIO*, error producido al grabar la voz; *ERROR_SERVER*, código de error enviado por el servidor; *ERROR_SPEECH_TIMEOUT*, producido cuando no se reconoce voz en un tiempo determinado.

RecognizerIntent, clase pública que da soporte al servicio de reconocimiento de voz. Para comenzar a escuchar al usuario y reconocer su voz, es necesario crear un *Intent* (en el apartado 3.5.2 se explica más concretamente qué son y cómo funcionan los *Intents*) o acción que permite activar el *SpeechRecognizer* que se encuentra fuera de la aplicación

de *Telegram*. Esta clase **RecognizerIntent** contiene las constantes necesarias para poder llevar a cabo esa acción. Por ejemplo, *ACTION_RECOGNIZE_SPEECH* inicia la actividad que solicita hablar al usuario y que lo enviará a los servidores de Google. Otras constantes necesarias son *EXTRA_LANGUAGE_MODEL* y *LANGUAGE_MODEL_FREE_FORM*, las cuales indican al reconocedor de voz qué modelo de habla y qué idioma se debe utilizar para el reconocimiento, de lo contrario no interpretaría los sonidos o fonemas de forma correcta.

RecognitionListener, interfaz que define los métodos de *callback* que reciben las notificaciones de lo que ocurre mientras se está realizando el reconocimiento de voz. El método *onReadyForSpeech()* es llamado cuando el reconocedor de voz está preparado para escuchar al usuario hablar y *onBeginningOfSpeech()* es llamado cuando se empieza a reconocer la voz del usuario.

El método *onError()* notifica el código de error que se ha producido al llevar a cabo el reconocimiento, en el caso de que se produjera. Una vez se ha reconocido una palabra u oración con éxito, el servidor devuelve los resultados obtenidos llamando al método *onResults(Bundle results)*.

La figura 3.3 muestra un breve diagrama que intenta clarificar lo explicado anteriormente. Cabe destacar que no es el diagrama UML de toda la API de Android *android.speech*, sino sólo la parte más relevante para este proyecto.

3.5. Elementos relevantes para el desarrollo en *Android*

Con el objetivo de modificar la aplicación de *Telegram* para la plataforma de Android, debemos conocer los aspectos y conceptos sobre desarrollo de aplicaciones en dicha plataforma. Es necesario tener claros los elementos que la componen y la funcionalidad de cada uno de ellos.

Las aplicaciones en Android están escritas en lenguaje *Java*. *Android SDK tools* compilan el código generando un *APK: Android Package* (.apk), este archivo contiene todo el contenido necesario para ejecutar la aplicación en el dispositivo.

Existen una serie de componentes esenciales para poder crear cualquier aplicación en Android. Cada componente existe como entidad propia, y algunos componentes dependen de otros, pero cada uno por separado juega un papel específico que ayuda a definir el comportamiento general de la aplicación. Cada uno de ellos tiene un ciclo de vida distinto que define cómo se crea y se destruye. A continuación, describiremos cada uno de estos componentes. [7, 8, 11, 22]

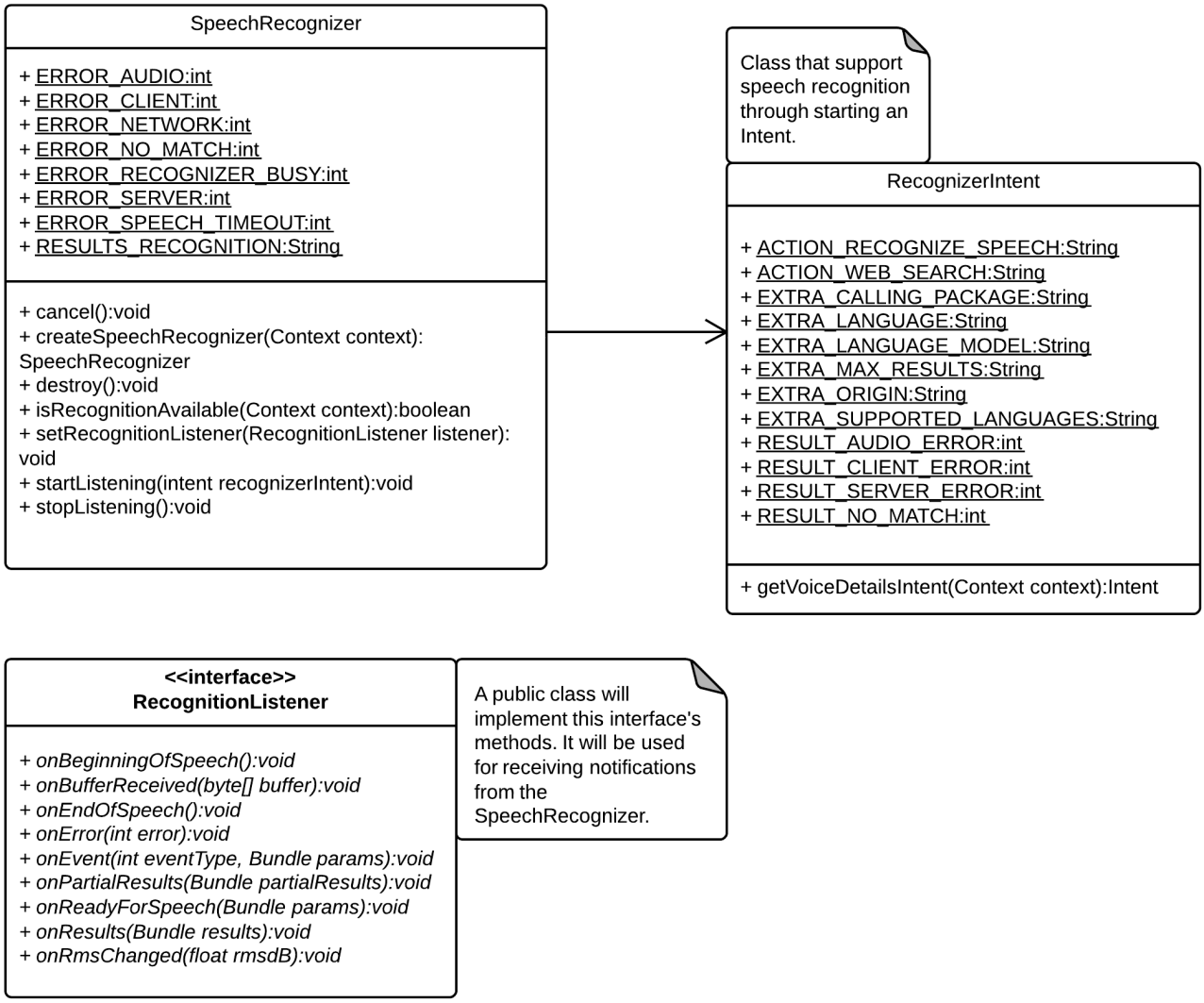

Figura 3.3: Diagrama UML *SpeechRecognizer*.

3.5.1. *Activities*

Una actividad se encarga de mostrar al usuario la interfaz gráfica con la que interactúa, y es el medio de comunicación entre la aplicación y el usuario. Se define una actividad por cada interfaz de la aplicación, o lo que es lo mismo, por cada ventana distinta que tenga en la aplicación. Cada actividad puede iniciar una nueva con el objetivo de realizar distintas acciones. Cada vez que una actividad comienza, la que se estaba ejecutando se detiene y el sistema la conserva en una pila, de esta manera cuando el usuario pulse el botón “atrás” podrá volver a la que se estaba ejecutando.

Las actividades tienen un ciclo de vida, es decir, pasan por diferentes estados desde que se inician hasta que se destruyen. Sus 3 posibles estados son:

- **Activo**: La actividad está en ejecución, en ese momento es la tarea principal.
- **Pausado**: La actividad se encuentra semi-suspendida, es decir, aún se está ejecutando y es visible, pero no es la tarea principal. Se debe guardar la información en este estado para prevenir una posible pérdida de datos en caso de que el sistema decida prescindir de ella para liberar memoria.
- **Parado**: La actividad está detenida, no es visible al usuario y el sistema puede liberar memoria. En caso de necesitarla de nuevo, será reiniciada desde el principio.

Cuando se pausa o detiene una actividad porque otra nueva se inicia, se notifica este cambio de estado a través de los métodos de *callback* del ciclo de vida de la actividad. Son varios los métodos que una actividad puede recibir debido a un cambio en su estado -si el sistema está creando la aplicación, parándola, reanudándola o destruyéndola- y cada *callback* ofrece la oportunidad de realizar el trabajo apropiado para cada cambio de estado. Por ejemplo, cuando se detiene, la actividad debe liberar objetos que ocupen mucha memoria, como conexiones de red o bases de datos. Cuando la actividad se reanuda, se puede volver a adquirir los recursos necesarios y reanudar las acciones que fueron interrumpidas. Estas transiciones de estado son parte del ciclo de vida de la actividad. Los métodos fundamentales de *callback* son los descritos a continuación.

OnCreate(Bundle savedInstanceState): Llamado cuando la actividad es creada por primera vez. Es el método que crea la actividad. Recibe un parámetro de tipo Bundle, que contiene el estado anterior de la actividad, para preservar la información que hubiera, en caso de que hubiera sido guardada. También puede iniciarse a null si la información anterior no es necesaria o no existe.

OnRestart(): Reinicia una actividad tras haber sido parada (si continúa en la pila de tareas). Se inicia desde cero.

OnStart(): Se ejecuta inmediatamente después de *onCreate(Bundle savedInstanceState)*, o de *onRestart()* según corresponda. Muestra al usuario la actividad. Si ésta va a estar en un primer plano, el siguiente método debe ser *onResume()*. Si por el contrario se desarrolla por debajo, el método siguiente será *onStop()*.

OnResume(): Establece el inicio de la interactividad entre el usuario y la aplicación. Solo se ejecuta cuando la actividad está en primer plano. Si necesita información previa, el método *onRestoreInstanceState()* aportará la situación en que estaba la actividad al llamar al *onResume()*. También puede guardar el estado con *onSaveInstanceState()*.

OnPause(): Se ejecuta cuando una actividad va a dejar de estar en primer plano, para dar paso a otra. Guarda la información, para poder restaurar cuando vuelva a estar activa en el método onSaveInstanceState(). Si la actividad vuelve a primer plano, el siguiente método será *onResume()*. En caso contrario, será *onStop()*.

OnStop(): La actividad pasa a un segundo plano por un largo período. Como ya se ha dicho, el sistema puede liberar el espacio que ocupa en caso de necesidad, o si la actividad lleva parada mucho tiempo.

OnDestroy(): Es el método final de la vida de una actividad. Se llama cuando ésta ya no es necesaria, o cuando se ha llamado al método *finish()*.

Además de estos métodos, cabe destacar dos más, que también son importantes:

OnSavedInstanceState(): Guarda el estado de una actividad. Es muy útil cuando se va a pausar una actividad para abrir otra.

OnRestoreInstanceState(): Restaura los datos guardados en *onSavedInstanceState()* al reiniciar una actividad.

La figura 3.4 resume el ciclo de vida de una actividad.

3.5.2. *Intents*

Las actividades proporcionan las partes reutilizables e intercambiables del flujo de componentes de interfaz de usuario en las aplicaciones de Android. Sin embargo, para que una actividad invoque a otra y se puedan comunicar entre ellas son necesarios los *Intents*, unidades de comunicación. Los *Intents* son el medio de activación de los componentes excepto de los *content providers*. Pueden ser explícitos o implícitos. Los implícitos no especifican el componente al que va destinado, mientras que el explícito sí. Aunque los *Intents* faciliten la comunicación entre los componentes de varias maneras, tiene tres usos principales:

- **Para empezar una actividad**: Dentro de una actividad se puede empezar una nueva instancia de otra actividad pasándole un *Intent* a *startActivity()*. Este *Intent* describe la actividad que da comienzo y posee los datos necesarios para iniciarla. Si se quiere recibir un resultado cuando finaliza la actividad, se debe llamar al método *starActivityForResult()*.
- **Para empezar un servicio**: Se puede comenzar un servicio para llevar a cabo una operación (como por ejemplo descargar un archivo) pasándole un *Intent* al método *startService()*.

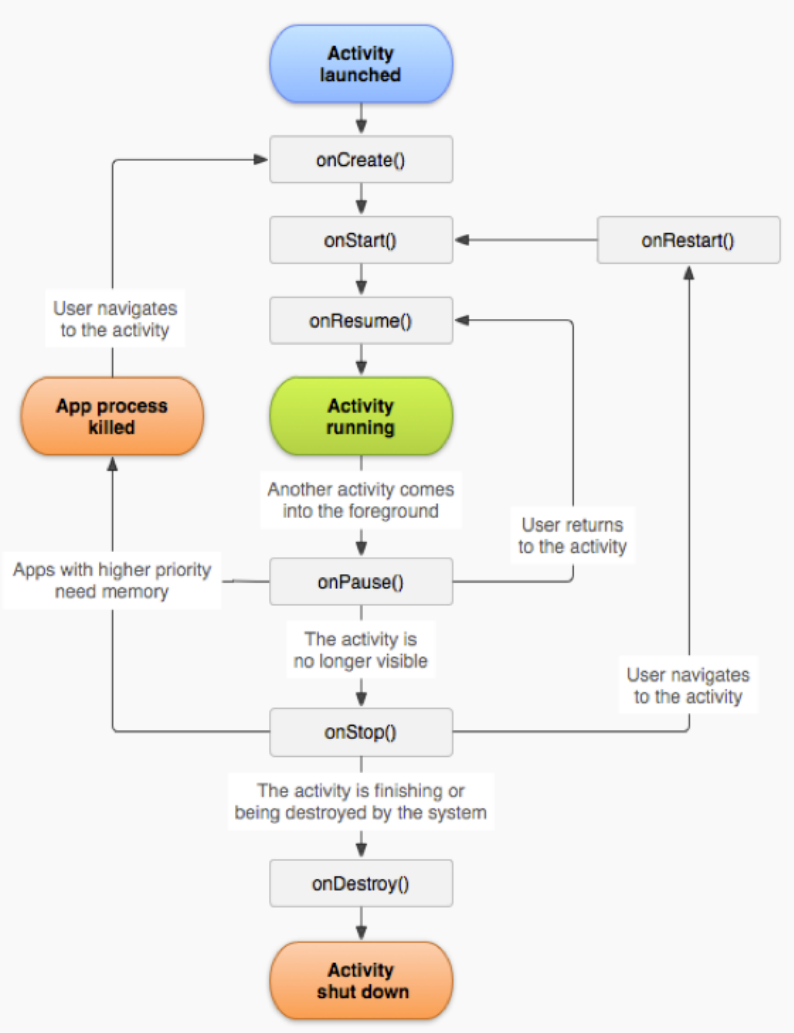

Figura 3.4: Ciclo de vida de la Actividad [1].

- **Para entregar una emisión o mensaje *broadcast***: Un mensaje de *broadcast* puede recibirlo cualquier aplicación.

3.5.3. *Services*

Los servicios (o *service*) son tareas no visibles que se ejecutan siempre por debajo, incluso cuando la actividad asociada no se encuentra en primer plano. Tiene un hilo propio (aunque no se pueden ejecutar solos), lo que permite llevar a cabo cualquier tarea, por pesada que sea. Por ejemplo, una aplicación reproductora de música será implementada como una servicio para poder seguir escuchando música mientras el usuario utiliza otras aplicaciones de su móvil.

Como las actividades, los servicios ofrecen métodos de *callback* que controlan su ciclo de vida.

3.5.4. *Content Providers*

Se encargan de que la aplicación pueda acceder a la información que necesita, siempre que se haya declarado el correspondiente *provider* en el *AndroidManifest.xml*, compartiendo información sin revelar estructura u orden interno.

3.5.5. *AndroidManifest*

Este fichero es un documento XML en el que se declaran los elementos de la aplicación, así como sus restricciones, permisos, procesos, acceso a datos e interacciones con elementos de otras aplicaciones. Cada elemento se declara con una etiqueta única. No debe confundirse este documento con el XML asociado a cada actividad.

Capítulo 4

Diseño e Implementación

En este capítulo mostraremos el diseño detallado del módulo de Control por Voz implementado. Comenzaremos explicando las modificaciones generales realizadas a Telegram así como las clases utilizadas para su implementación, y las vinculaciones que existen entre ellas. Posteriormente, explicaremos en detalle tanto la codificación del sistema de síntesis de voz *TextToSpeech*, como del reconocimiento de voz *SpeechRecognizer*.

4.1. Modificaciones realizadas a *Telegram*

En primer lugar, y antes de comenzar con la explicación detallada de la implementación, describimos el funcionamiento general del módulo de Control por Voz a través de un diagrama de actividad o de flujo (Figura 4.1). Dividiremos el diagrama en pequeños pasos que se irán explicando para intentar clarificarlo lo máximo posible. De esta manera, se intenta facilitar el posterior seguimiento de lo que se detalle y explique sobre las distintas partes del módulo implementadas. Cabe destacar que el diagrama de actividad presenta el desarrollo de forma cíclica pues los comandos intentan reutilizar pasos, de manera que no se repiten partes de código en la implementación. Eso quiere decir que al terminar un paso se vuelve a empezar el diagrama de actividad y dependiendo del comando se actúa de una determinada forma. Este diseño cíclico se detallará más adelante.

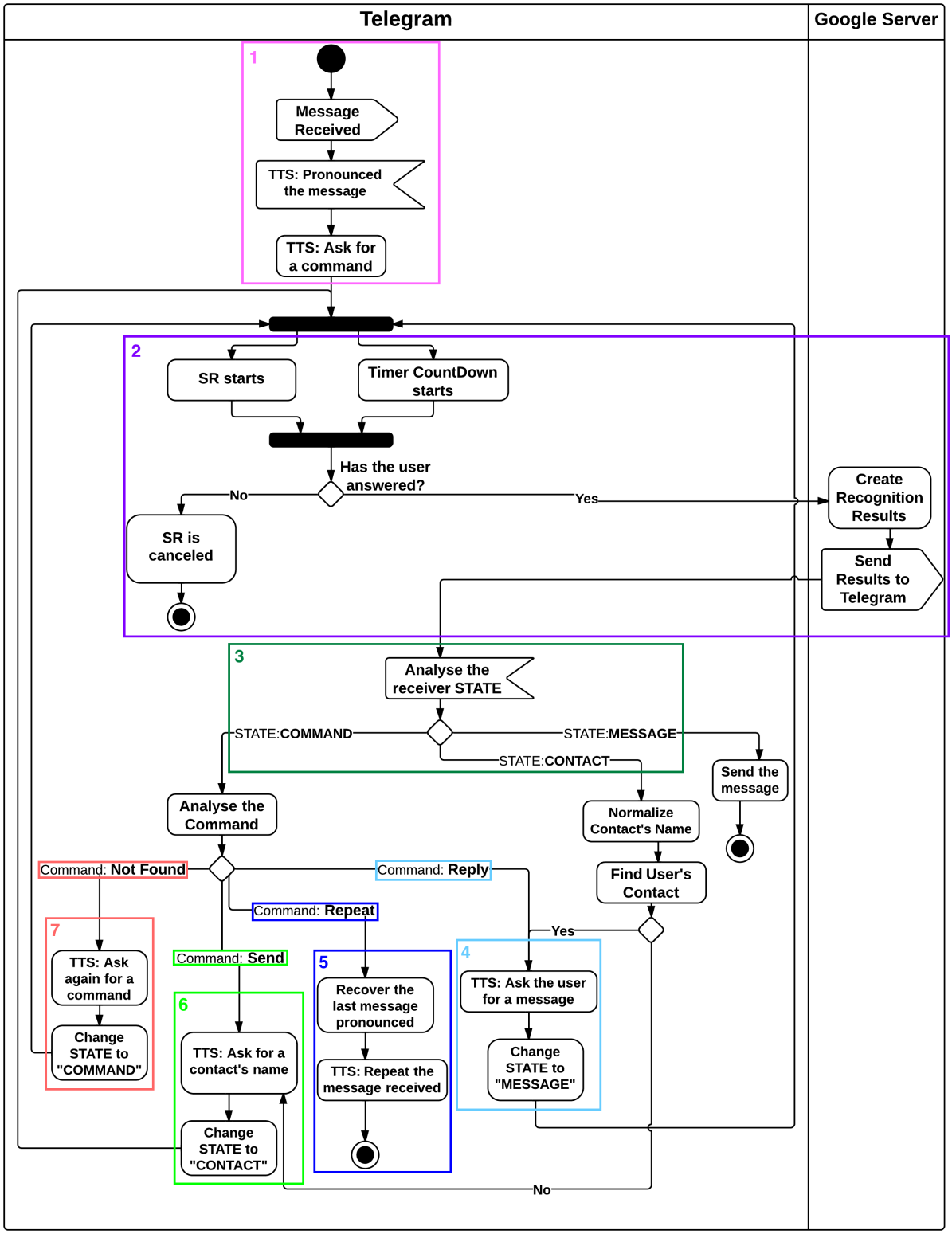

Figura 4.1: Diagrama de Actividad del módulo de Control por Voz.

Paso 1

El módulo de Control por Voz se inicia o activa con la recepción de un mensaje nuevo en la aplicación de *Telegram*. Una vez recibido un mensaje se inicia el *TextToSpeech* (TTS) y se produce la síntesis del mensaje por voz. Se pronunciarán tanto el mensaje, como el nombre del contacto que lo envió. Al finalizar la síntesis del mensaje, la aplicación preguntará qué se desea realizar ("Diga un comando *Telegram*").

Paso 2

A partir de este momento se activará el *SpeechRecognizer* (SR), y a la vez comenzará una cuenta atrás o *timer* de 5 segundos. En este periodo de tiempo se pueden dar dos casos: 1. El usuario no dice nada, por lo que no se lleva a cabo ningún reconocimiento de voz y el *SpeechRecognizer* se cancela. 2. El usuario pronuncia un comando: "repetir" , "enviar", "responder" o ninguno de los anteriores. Aquello que se pronuncie será enviado directamente a los servidores de Google, los cuales se encargarán de recoger los resultados obtenidos. Una vez esto ocurra, los servidores enviarán a la aplicación los resultados.

Paso 3

En este paso se debe analizar el estado del módulo en ese instante. Existen tres posibles estados: "COMMAND", "CONTACT" y "MESSAGE". En un primer momento el estado será "COMMAND", pues lo que se espera recibir es uno de los tres comandos posibles. Por ello, al encontrarse en ese estado pasaremos a analizar el comando recibido. Posteriormente, los estados se irán modificando dependiendo del comando que se esté ejecutando. En el caso en el que el estado sea "CONTACT", lo siguiente que la aplicación esperará reconocer por parte del usuario será el nombre de un contacto de *Telegram*. Si el módulo no consigue encontrar el contacto en la agenda, se volverá a solicitar al usuario que diga de nuevo un contacto. Finalmente, si el estado es "MESSAGE", la aplicación esperaría reconocer un mensaje para enviar o bien, recuperar el último mensaje recibido para volver a pronunciarlo puesto que no se ha oído o entendido la síntesis de voz.

Paso 4

Siempre que el módulo se encuentre en el estado "COMMAND", analizaremos de qué comando se trata. Cuando se trate del comando "responder" (reply), el TTS se ejecutará de nuevo para poder preguntar al usuario cuál es el mensaje que desea enviar ("Diga el mensaje"). A continuación, modificaremos el estado del módulo a "MESSAGE" pues volverá a ejecutarse el módulo de nuevo comenzando por el paso 2, en el que se activan de nuevo el SR y el *timer*.

Paso 5

En el caso en el que el comando sea "repetir" (repeat), el módulo se debe encargar de recuperar el último mensaje que se haya recibido en la aplicación. Una vez obtenido, volverá a activarse el TTS para sintetizar por voz el mensaje y una vez pronunciado, el módulo se desactiva.

Paso 6

Si se reconoce el comando "enviar" (send), el TTS se activará de nuevo para preguntar a quién se desea enviar el mensaje ("Diga un contacto de de *Telegram*"). A continuación, se modifica el estado a "CONTACT" y se reanuda de nuevo todo el proceso a partir del paso 2. En esta iteración del proceso el usuario deberá decir el nombre de un contacto de *Telegram* al que enviar el proceso. La aplicación se encargará de buscarlo en la agenda y se procederá a realizar el paso 4, en el que la aplicación se encarga de preguntar al usuario por el mensaje a enviar.

Paso 7

Si el módulo no reconoce ninguno de los tres posibles comandos (command Not Found), se activará de nuevo el TTS para preguntar de nuevo por un comando ("Diga un comando *Telegram*") y se modificará el estado a "COMMAND", retomando de nuevo el proceso a partir del paso 2.

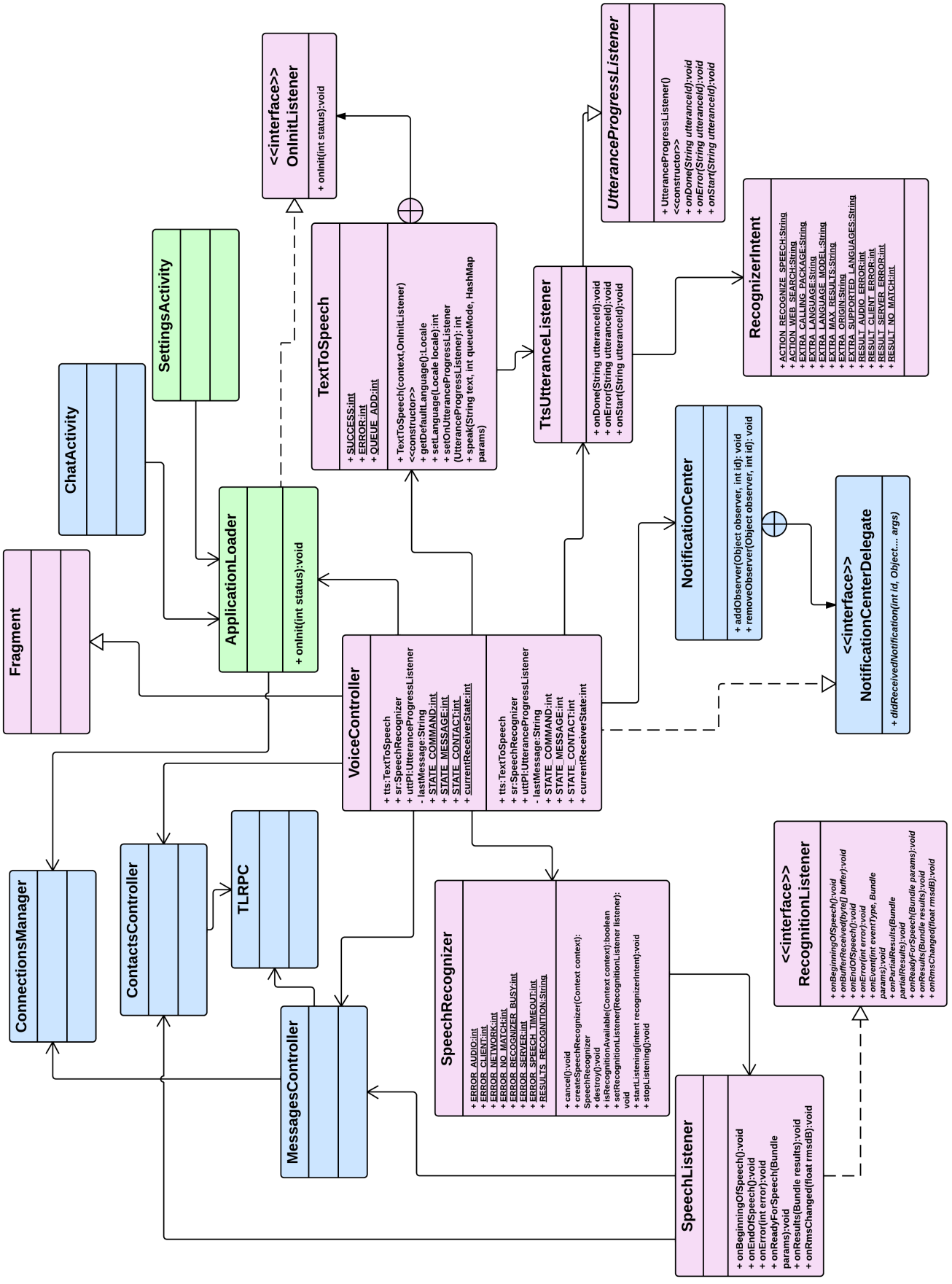

Figura 4.2: Diagrama UML de clases.

Una vez descrito el diagrama de actividad del módulo de Control por Voz, presentamos el diagrama UML de clases en la figura 4.2. Las clases implementadas por nosotros se encuentran coloreadas en rosa. Las clases propiamente de *Telegram* que han sido modificadas, se encuentran coloreadas en verde. Finalmente, están coloreadas en azul las clases también pertenecientes a *Telegram*, que hemos tenido que analizar y de las cuales hemos sido dependientes para desarrollar el módulo implementado. Describiremos y mostraremos la interacción entre clases a través de diversos diagramas UML [21] de secuencia correspondientes a los casos de uso más relevantes.

ApplicationLoader

Clase en la que se crean e inicializan tanto el *TextToSpeech* como el *SpeechRecognizer*. En esta clase se implementa el método *onInit()*, que se encuentra definido en la interfaz *OnInitListener* de la librería del *TextToSpeech*. Este método se encarga de asegurar que se puede iniciar el motor o *Engine* de *TextToSpech* en el dispositivo móvil. En el caso en el que el TextToSpeech no esté instalado en el dispositivo, accederá al *Play Store* para comenzar la descarga de los archivos necesarios para su uso. En la figura 4.3 mostramos el diagrama de secuencia de la inicialización de *Telegram* a partir de la clase *ApplicationLoader*.

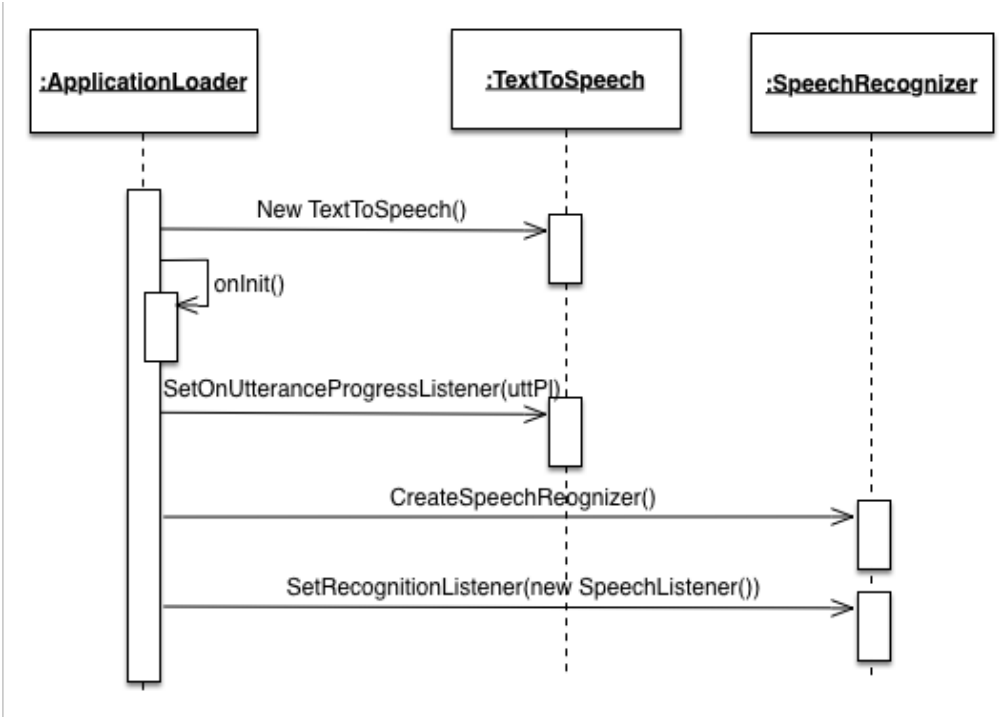

Figura 4.3: Caso de uso: Inicialización de *Telegram*.

VoiceController

La clase *VoiceController* implementa la mayor parte del módulo de Control por Voz. Se decidió crear esta clase para no tener que modificar directamente el código fuente de *Telegram* y que de esta manera la interdependencia fuera la menor posible, minimizando así el acoplamiento. De la misma forma, con la creación de esta clase se continúa con el patrón que sigue la implementación de *Telegram*. Al igual que existen las clases *MessagesController* o *ContactsController* para controlar sus respectivos módulos, nosotros intentamos hacer lo mismo con *VoiceController*.

En esta clase se definen los atributos relacionados con la codificación del *TextToSpeech* y del *SpeechRecognizer*. Además, se definen también las estructuras de datos *languageDependentPrefix* y *languageDependentCommands*. Éstas se encargan de asignar el idioma correcto al módulo de Control por Voz dependiendo del idioma en el que esté configurado el dispositivo móvil. La configuración del idioma se puede conseguir con la *Locale*, la cual se encarga de presentar la información (idioma, fechas, horarios) adaptándose a las propiedades que el usuario haya previamente configurado en su dispositivo móvil. De esta manera la síntesis de voz puede ser extendida para múltiples idiomas, y actualmente se encuentra implementado para Castellano, Inglés o Alemán, como se muestra en las siguientes líneas del código fuente. En la séptima línea de código se muestra como primero se consulta el idioma que posee el dispositivo móvil, si el idioma es español (“es”) se pronunciará “Mensaje enviado por” y se indexará con el nombre del remitente.

```
1 Map<String,String> languageDependentPrefix = new HashMap<String,
     String>(){
2    {put("es","Mensaje enviado por ");
3     put("en","Message sent by ");
4     put("de","Nachricht per ");
5     }
6 };
7 message = languageDependentPrefix.get(Locale.getDefault().
    getLanguage()) + userFrom.first_name + " " + userFrom.
    last_name;
```

Otras estructuras que caben destacar son *ttsWithID* y *userToAnswer*. La estructura *ttsWithID* es un *HashMap* que nos permite asignar un identificador a cada mensaje, dicho identificador es requerido por el *TtsUtteranceListener*, de tal modo que dependiendo de este, indica si se debe arrancar el SpeechRecognizer o no (esto se detallará en el apartado 4.3). La estructura *userToAnswer* actúa como cola FIFO(*FirstInFirstOut*) y se encarga de almacenar el *id* del último usuario que ha enviado el mensaje. Se decidió crear esta estructura FIFO pues en el caso de que se reciban varios mensajes, al procesarse

en orden de recepción, se deben recuperar los remitentes en ese mismo orden. Así, podremos acceder al *id* del usuario que ha enviado el mensaje para poder responderle. Para consultar y modificar esta estructura de datos utilizaremos los métodos *getFirstUserId()* y *setFirstUserId()*.

Esta clase implementa la interfaz *NotificationCenterDelegate* a través del método *didReceiveNotification()*. En primer lugar, nos debemos registrar en la lista de subscriptores a través del método *addObserver()*. Así, cada vez que se produzca un evento que posea como identificador *didReceivedNewMessages* se ejecutará el método *didReceiveNotification()*, es decir, este método se ejecutará cada vez que el módulo esté activado y se reciba un mensaje nuevo en la aplicación de *Telegram*. Esto queda representado en el diagrama de secuencia de la figura 4.4. Además, inicializaremos la clase *TtsUtteranceListener* necesaria cuando utilicemos el *TextToSpeech* (esto se explica detalladamente en el apartado 4.2).

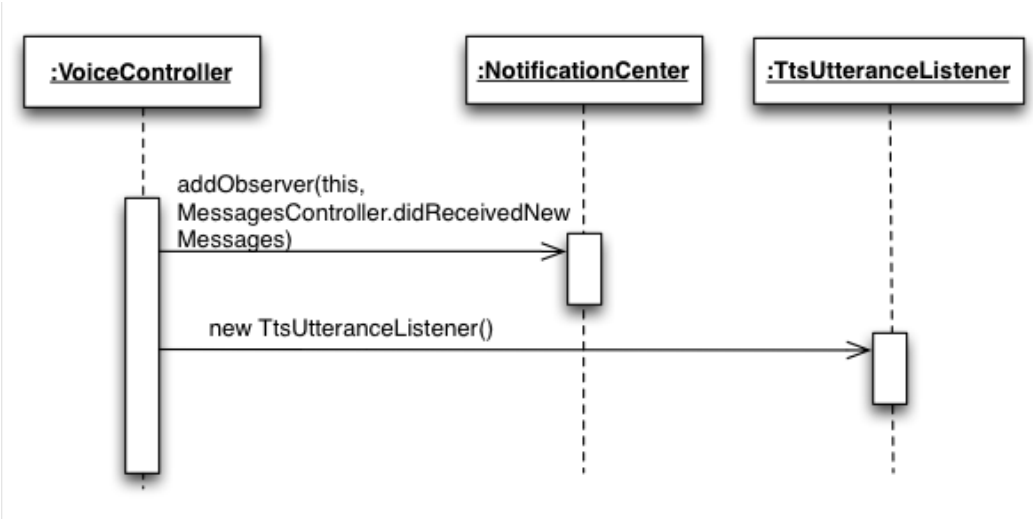

Figura 4.4: Caso de uso: Registro al *NotificationCenter*.

Otro método implementado en esta clase es *pronounceString()*. Este método se encarga de asegurar la correcta inicialización del *TextToSpeech* tras la ejecución de la clase *ApplicationLoader*, y en el caso de que así sea, se procederá a pronunciar el mensaje que se haya recogido en el método *didReceivedNotification()*. Además, es en este método donde se actualiza la estructura de datos *ttsWithID*, descrita anteriormente. Definimos además las constantes y métodos necesarios para conocer el estado en el que se encuentre el módulo, esta parte se explicará con detalle en el apartado 4.3.

En la figura 4.5 se muestra el diagrama UML de la clase *VoiceController* para detallar los atributos, constantes y métodos utilizados.

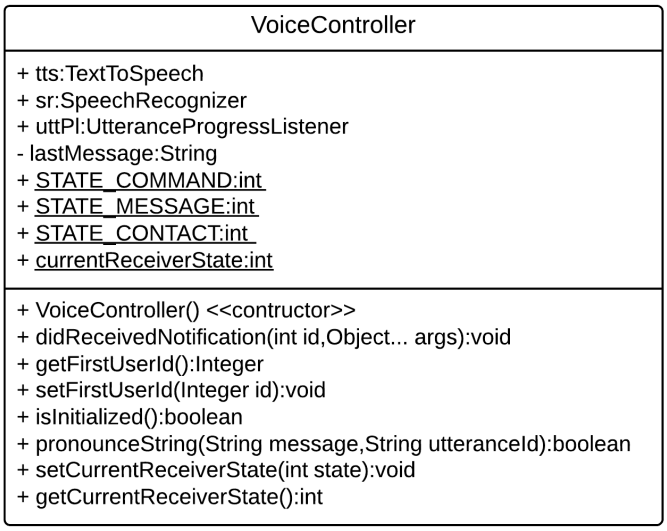

Figura 4.5: Diagrama UML de la clase *VoiceController*.

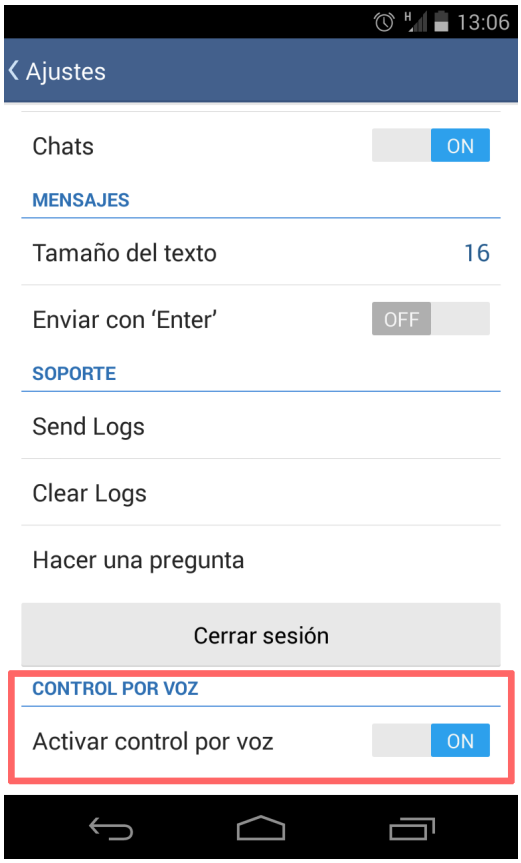

Figura 4.6: Interfaz de "Ajustes" de *Telegram* con activación del Control por Voz.

SettingsActivity

Clase modificada para añadir en la interfaz de "Ajustes" de *Telegram* la posibilidad para el usuario de poder activar o desactivar el módulo de Control por Voz cuando lo desee. La figura 4.6 muestra como quedaría la interfaz tras su modificación.

NotificationCenter

Clase que define la interfaz *NotificationCenterDelegate*. Esta interfaz a su vez define el método *didReceiveNotification()* y que es implementado en la clase *VoiceController* como ya hemos explicado anteriormente.

MessagesController

Esta clase implementa el método *sendMessage()* necesario para poder enviar los mensajes que sean pronunciados por voz. A través de ella podemos acceder al usuario que ha enviado el mensaje, de manera que recuperamos tanto el nombre como el apellido para poder sintetizarlos a través del *TextToSpeech*.

ChatActivity

Gracias a esta clase la interfaz de los chat queda actualizada. Así, permite que cada mensaje que se pronuncie por voz y se envíe quede registrado en el chat en el que se esté manteniendo o en el que se haya iniciado la conversación.

ContactsController y TLRPC

A la hora de enviar un mensaje a un contacto determinado de *Telegram* elegido por el usuario necesitamos acceder a toda la agenda que este posee. Gracias a estas dos clases podemos acceder a ellos y enviar el mensaje al contacto que el usuario haya pronunciado por voz.

En el diagrama de clases también cabe destacar las clases **TextToSpeech**, **TtsUtteranceListener**, **SpeechRecognizer** y **SpeechListener** las cuales serán explicadas en detalle en los próximos apartados.

4.2. Codificación TextToSpeech

Como ya hemos comentado anteriormente, una vez inicializado el motor de *TextToSpeech* y tras haber recibido un mensaje en la aplicación, se produce su síntesis de voz gracias al método *speak()*. Sin embargo, para detectar cuándo la síntesis de voz ha terminado, y cuándo debemos comenzar el reconocimiento de voz, debemos crear la clase ***TtsUtteranceListener*** que se encargue de implementar los métodos de *callback* que define la clase abstracta *UtteranceProgressListener*: *onDone(String utteranceId), onError(String utteranceId) y onStart(String utteranceId)*. En nuestro caso destacamos el método *OnDone(String utteranceId)*, utilizado para resolver el problema que acabamos de mencionar y poder saber cuándo llevar a cabo el reconocimiento de voz. Es decir, debemos asegurarnos que se ha reproducido todo el mensaje recibido antes de utilizar el *SpeechRecognizer()* que comenzará a escuchar la respuesta que vaya a dar el usuario.

Para que se llame a estos eventos de *callback* implementados por la clase *TtsUtteranceListener*, se debe establecer cuál es la clase que los implementa a través del método *setOnUtteranceProgressListener(uttPL)*, que se ejecuta en la clase principal *ApplicationLoader*, siendo *uttPl* el objeto que referencia a la clase *TtsUtteranceListener* (la figura 4.3 que hace referencia a la inicialización de *Telegram*, muestra lo que se acaba de explicar). Estos métodos de *callback* serán llamados directamente por el TextToSpeech en cuanto la referencia al objeto de *TtsUtteranceListener* sea distinta de *null*. Esto se muestra en el diagrama de secuencias de la figura 4.7.

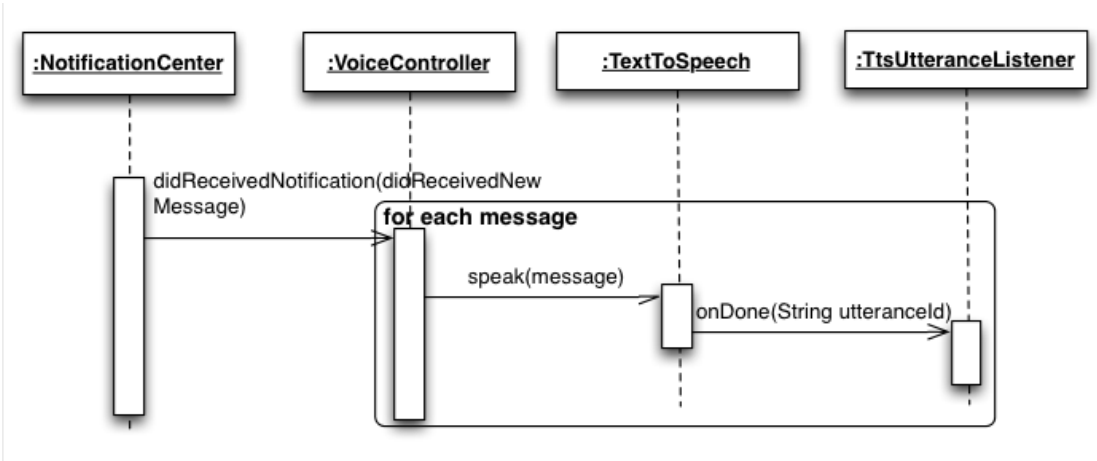

Figura 4.7: Caso de uso: Procesar eventos con *id didReceivedNewMessages*.

Como acabamos de explicar, el método *OnDone(String utteranceId)* siempre será llamado en cuanto exista la referencia a la clase *TtsUtternaceListener*. Sin embargo, sólo se ejecutará cuando su parámetro *utteranceId* contenga la cadena de caracteres

"command". El método *OnDone(String utteranceId)* es el encargado de iniciar el *SpeechRecognizer*, pero no queremos que se reconozca la voz del usuario cada vez que se pronuncia o se sintetiza un mensaje, sino sólo y únicamente cuando sea necesario y el módulo necesite una respuesta por parte del usuario. Para distinguir cuando se debe comenzar el reconocimiento de voz utilizaremos como *utteranceId* la cadena "command" que se pasará como parámetro en el método *pronounceString(String message,String utteranceId)*, concretamente, *pronounceString(messageToPronounce,"command")*.

La implementación del método *OnDone(String utteranceId)* se lleva a cabo de la manera que muestra el diagrama de la figura 4.8. El método *startLlistening()* de la librería *SpeechRecognizer* sólo puede ejecutarse en el hilo principal de la aplicación y este hilo lo conseguimos accediendo al *Looper* (hilo utilizado para enviar las llamadas a los componentes principales de la aplicación: actividades, servicios...) del contexto de la aplicación y creando un manejador (clase *Handler* de *Android*) para dicho hilo.

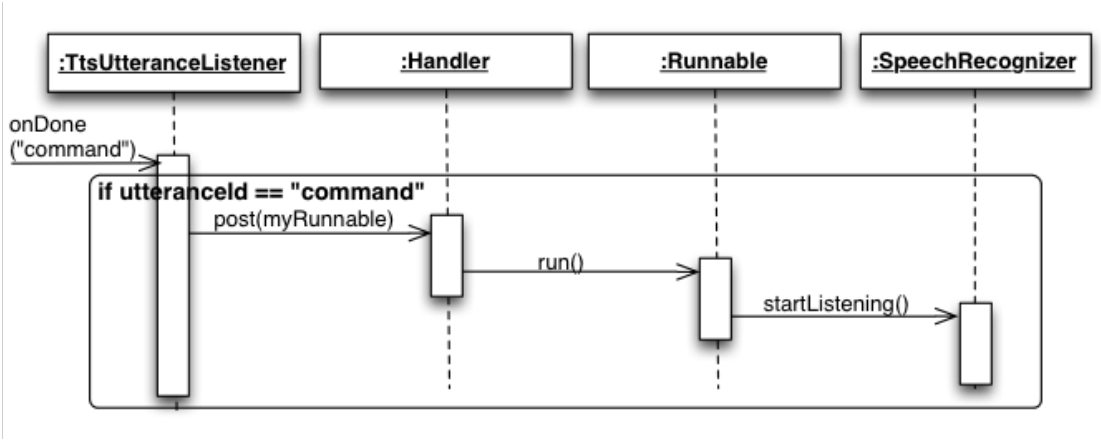

Figura 4.8: Implementación método *OnDone("command")*.Comienzo *SpeechRecognizer*.

4.3. Codificación SpeechRecognizer

Al igual que ocurre con la parte de *TextToSpeech*, a la hora de realizar el reconocimiento de voz con la librería *SpeechRecognizer* se debe crear una clase, en nuestro caso llamada *SpeechListener*, implementando los métodos de *callback* que define la interfaz *RecognitionListener*. Nosotros nos centramos en varios de esos métodos: *onReadyForSpeech(Bundle params), onBeginningOfSpeech() y onResults(Bundle results)*.

El reconocimiento de voz comienza con la ejecución del método *startListening(Intent)* tras haberse finalizado la síntesis de voz, como ya hemos mencionado anteriormente. Es necesaria la creación de un *Intent* (3.5.2) pues se debe comenzar una nueva actividad

que sea capaz de lanzar el motor de reconocimiento de voz e interactuar con el servidor de Google, pues es este último el que se encarga de obtener y recoger los resultados y enviarlos de nuevo a la aplicación. Sin embargo, no siempre se obtienen resultados, puede darse el caso en el que el usuario no diga nada ya sea porque no es necesario o porque no quiere utilizar el módulo para envíar un mensaje. Para identificar este caso debemos utilizar los métodos de *callback*. El método *onReadyForSpeech(Bundle params)* se ejecuta justo cuando se lanza el reconocedor de voz (es identificable pues activa una señal sonora para avisar al usuario de que puede comenzar a hablar cuando la señal finalice). Es aquí donde comenzará una cuenta atrás o *timer* de 5 segundos, capaz de desactivar el reconocedor de voz si no se ha obtenido ningún resultado porque el usuario no ha pronunciado ninguna palabra en ese intervalo de 5 segundos. Esto se muestra en el diagrama de secuencia de la figura 4.9.

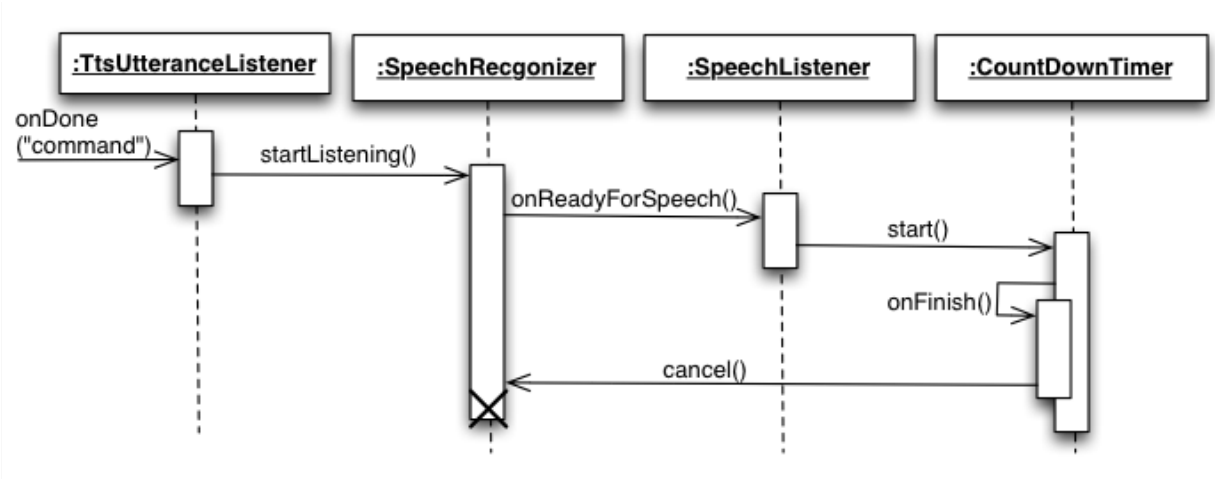

Figura 4.9: Cancelación del *SpeechRecognizer* al no recibir respuesta.

De igual forma, el método *onBeginningOfSpeech()* se ejecuta cuando el usuario comienza a hablar, y por tanto, cuando se empiezan a reconocer palabras. Por ello, al activarse y ejecutarse este método se deberá cancelar el *timer* para que el reconocedor de voz no se desactive mientras el usuario esté hablando. Una vez se ha reconocido el texto pronunciado por el usuario, éste se envía a los servidores de Google que crearán un resultado (*Bundle results*) y lo devolverán a la aplicación. Cuando se hayan recibido con éxito los resultados se activará el método de *callback onResults(Bundle results)*.

Nuestro método *onResults(Bundle results)* implementa una máquina de estados finitos analizando el resultado obtenido. De esta manera podemos distinguir los tipos de mensajes que vamos a reconocer y así saber qué debemos hacer en cada momento. Poseemos tres estados:

STATE_COMMAND. Cuando nos encontramos en este estado sabemos que el

siguiente paso será reconocer uno de los tres posibles comandos, “Enviar”, “Repetir”, “Responder” o en el caso de que no sea ninguno de estos se procedería a dar un error de comando pues sólo se esperan los tres que acabamos de describir.

STATE_CONTACT. Estado con el que se espera la recepción del nombre de un contacto de *Telegram* al que deseamos enviar un mensaje.

STATE_MESSAGE. Con este estado se espera procesar un mensaje, bien sea el último mensaje recibido o el mensaje que el usuario desee enviar.

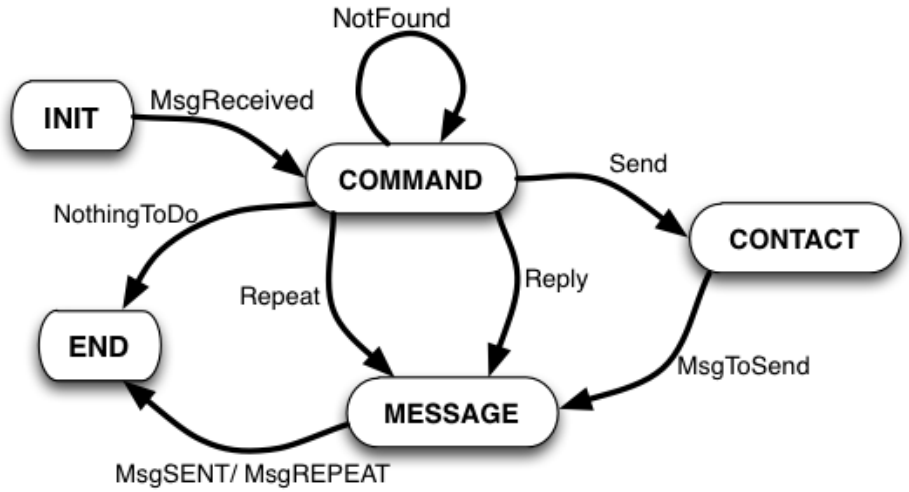

Figura 4.10: Diagrama de transiciones de la máquina de estados.

La figura 4.10 muestra la transición que se produce entre los distintos estados. Al recibir un mensaje pasaremos del estado inicial a *STATE_COMMAND*, el módulo espera que el usuario pronuncie un comando. En el caso de que sea el comando “Enviar”, pasaremos al estado *STATE_CONTACT*, pues el siguiente texto que esperamos recibir pronunciado por parte del usuario es el nombre de un contacto de *Telegram*. Cabe destacar que para la realización de este paso, hemos tenido que implementar el método *normalizeString()*, el cual se encarga de sustituir los caracteres especiales (como letras con tildes o diéreis) por caracteres fácilmente reconocibles. De esta manera existirá menos margen de error a la hora de reconocer el nombre del contacto. Un ejemplo de uso de dicho método sería al pronunciar el nombre de contacto “Raúl Sánchez”. Tras realizar varias pruebas hemos comprobado que el SR reconoce una o las dos palabras sin tildes, y el usuario a su vez puede tener el contacto almacenado en la agenda con o sin tildes. Por este motivo hace falta normalizar ambas cadenas de texto. En el caso de que el nombre del contacto no se reconozca se informará al usuario y se volverá a pedir el nombre. A continuación, mostramos el código fuente del método *normalizeString()*.

```
1     private String normalizeString(String input) {
2         // Cadena de caracteres original a sustituir.
3         String original = "áàäéèëíìïóòöúùuñÁÀÄÉÈËÍÌÏÓÒÖÚÙÜÑçÇy";
4         // Cadena de caracteres ASCII que reemplazarán los
              originales.
5         String ascii = "aaaeeeiiiooouuunAAAEEEIIIOOOUUUNcCi";
6         String output = input;
7         for (int i=0; i<original.length(); i++) {
8             // Reemplazamos los caracteres especiales.
9             output = output.replace(original.charAt(i), ascii.
                  charAt(i));
10        }//for i
11        return output;
12    }
```

Una vez reconocido y localizado en la agenda de *Telegram* el contacto al que enviar el mensaje modificamos el estado a *STATE_MESSAGE* y esperamos a recibir el mensaje que el usuario desea enviar. En el caso de que el comando sea “Repetir” modificaremos el estado a *STATE_MESSAGE* pues debemos recuperar el mensaje que se había recibido y volver a reproducirlo a partir del *TextToSpeech*. En el caso de que el comando recibido sea “Responder”, el estado pasará a ser *STATE_MESSAGE* y esperaremos a recibir el mensaje que el usuario desea enviar en forma de respuesta. Finalmente, si el comando no es correcto, bien porque no se ha reconocido o bien porque el usuario no sabe que opciones hay, no modificaremos el estado que continuaría siendo *STATE_COMMAND* y volvemos a preguntar al usuario qué desea hacer. En el caso en el que no diga nada y el *timer* haya expirado, el módulo de reconocimiento de voz se cancela.

A continuación, mostramos los diagramas de secuencias de los casos de uso correspondientes a cada uno de los posibles comandos que el usuario puede pronunciar. Cabe mencionar, que para simplificar los diagramas de secuencia las cadenas de texto se han escrito en castellano, sin embargo, ya hemos mencionado que el idioma de estas cadenas dependería de la configuración del dispositivo móvil.

La figura 4.11 muestra el diagrama de secuencia para el caso de uso en el que el usuario no haya escuchado bien el mensaje recibido y desee escucharlo de nuevo a través del comando "repetir". En él se puede observar como se cancela el *timer* al comenzar el reconocimiento de voz. A continuación, en el método *onResults()* se modifica el estado a *STATE_MESSAGE* cuando ha reconocido el comando "repetir", recuperaría el último mensaje recibido y llamaría de nuevo al *TextToSpeech* para pronunciarlo.

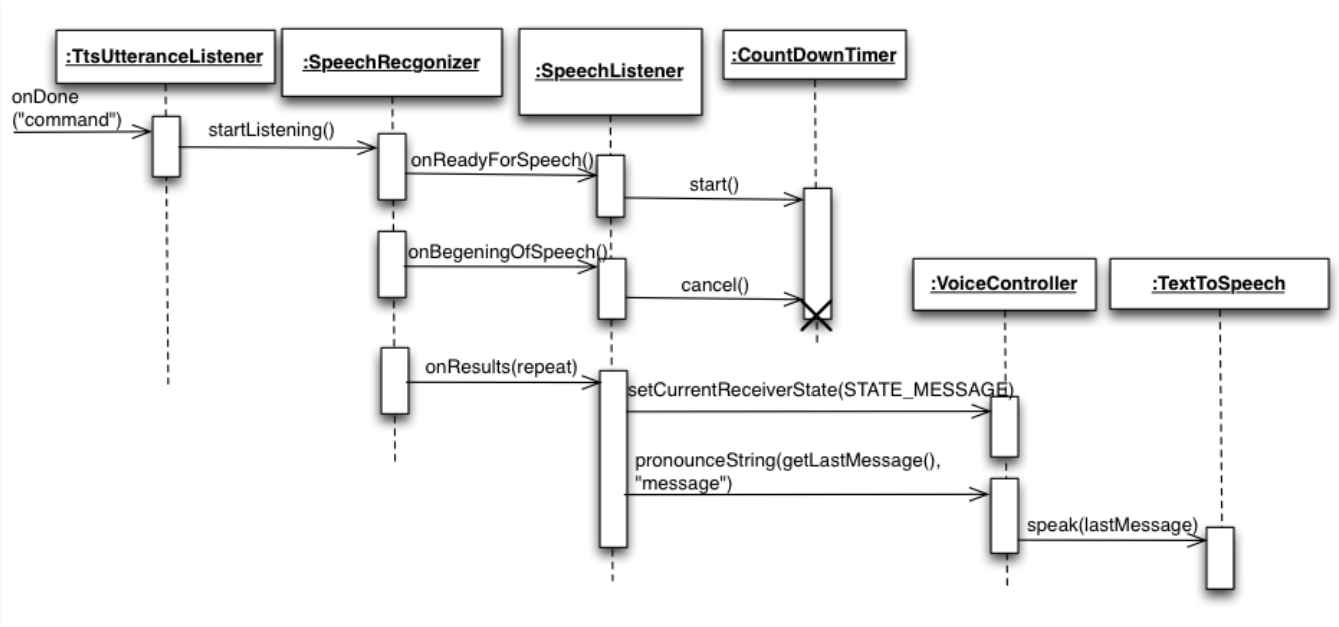

Figura 4.11: Diagrama de secuencia del caso de uso para el comando "Repetir".

La figura 4.12 muestra el primer paso del diagrama de secuencia del caso de uso en el que el usuario desee responder directamente al mensaje que acaba de escuchar. En este primer paso, al haber recibido la aplicación el comando "responder" lo siguiente que hace es preguntar al usuario qué mensaje desea enviar activando de nuevo el *TextToSpeech*, a la vez que modifica de nuevo el estado a *STATE_MESSAGE*. En la figura 4.13 se muestra el segundo paso en el que el usuario finalmente pronuncia el mensaje en sí que se desea enviar. A continuación, el *SpeechListener* se encarga de recuperar el *id* del usuario al que se desea responder y se procede a enviar dicho mensaje.

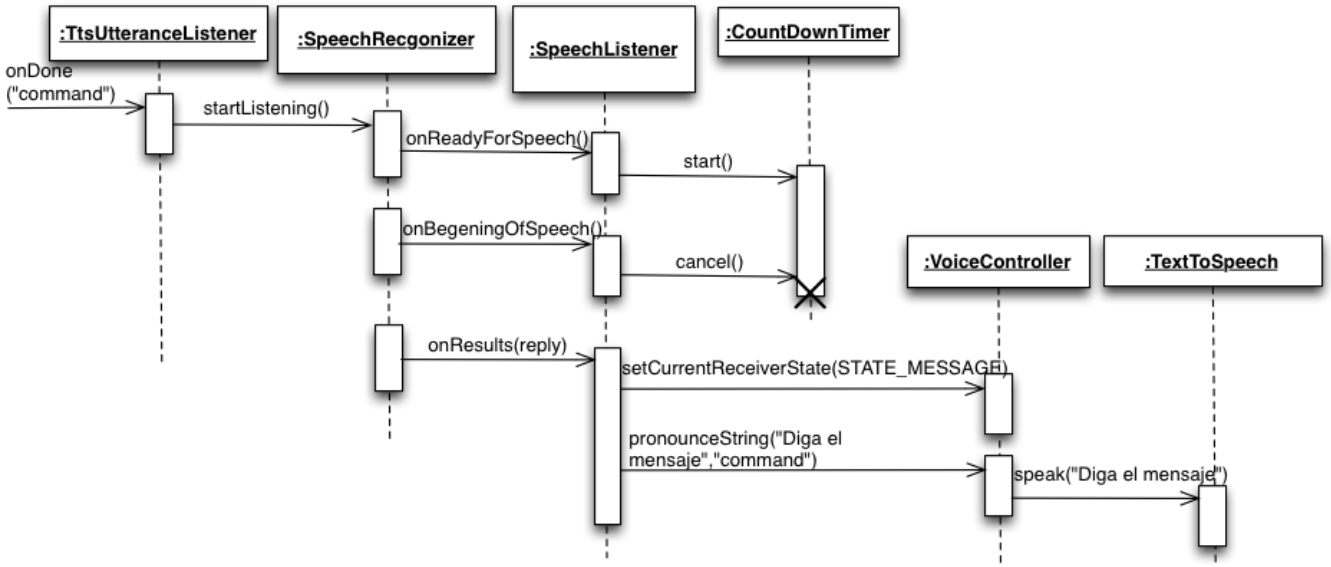

Figura 4.12: Paso 1 del diagrama de secuencia del caso de uso para el comando "Responder".

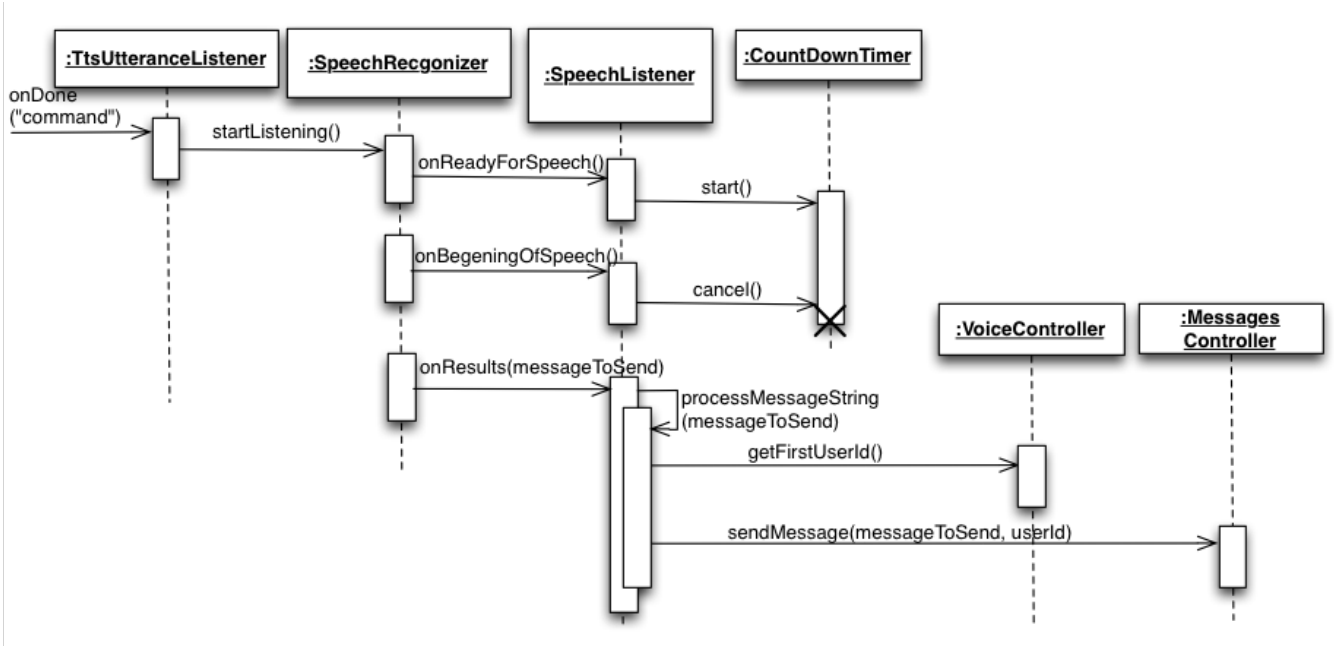

Figura 4.13: Paso 2 del diagrama de secuencia del caso de uso para el comando "Responder".

La figura 4.14 muestra el diagrama de secuencia del primer paso del caso de uso en que el usuario desea enviar un mensaje a cualquier contacto de *Telegram*. Tras haber reconocido el comando "enviar", se debe preguntar al usuario a qué contacto desea enviar el mensaje. El segundo paso se muestra en la figura 4.15, donde la aplicación se encarga de buscar el contacto de *Telegram*. Para ello se "normalizarán" tanto el nombre reconocido por voz, como los nombres de los contactos de la agenda, sustituyendo las letras por caracteres normales (quitando las tildes o acentos que puedan tener) y se procederá a hacer una comparación. Una vez encontrado, el módulo llevará a cabo los pasos del caso de uso "responder" explicado en la figura 4.12 y en la figura 4.13, donde se pide al usuario que pronuncie el mensaje a enviar.

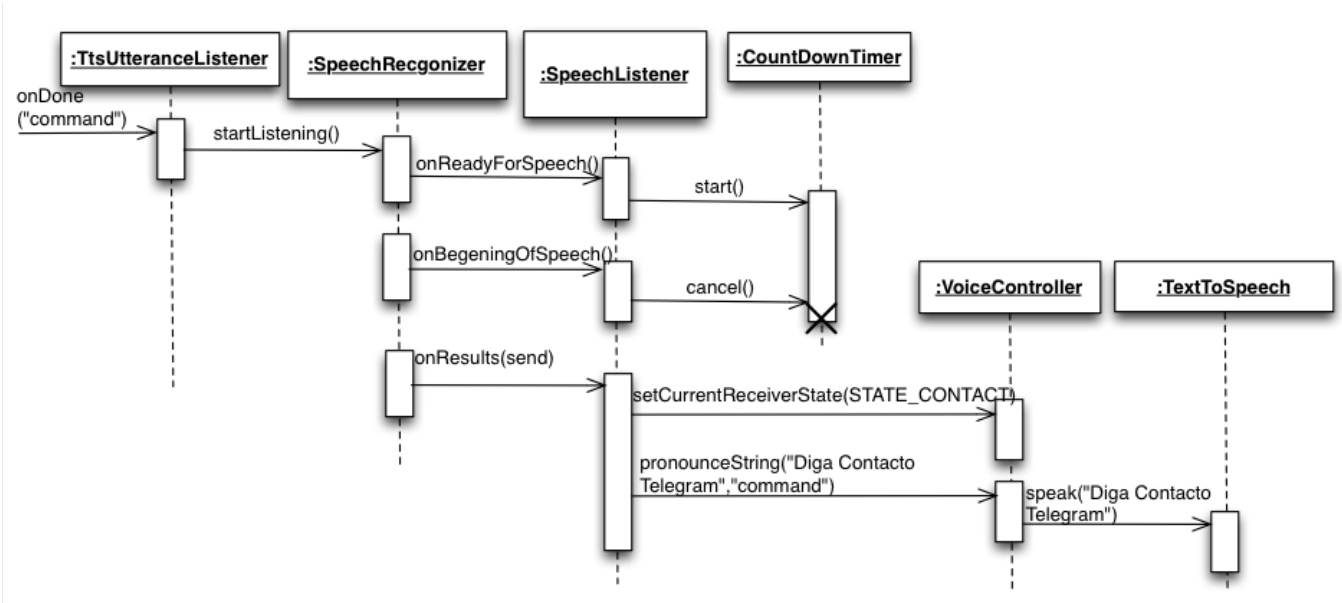

Figura 4.14: Paso 1 del diagrama de secuencia del caso de uso para el comando "Enviar".

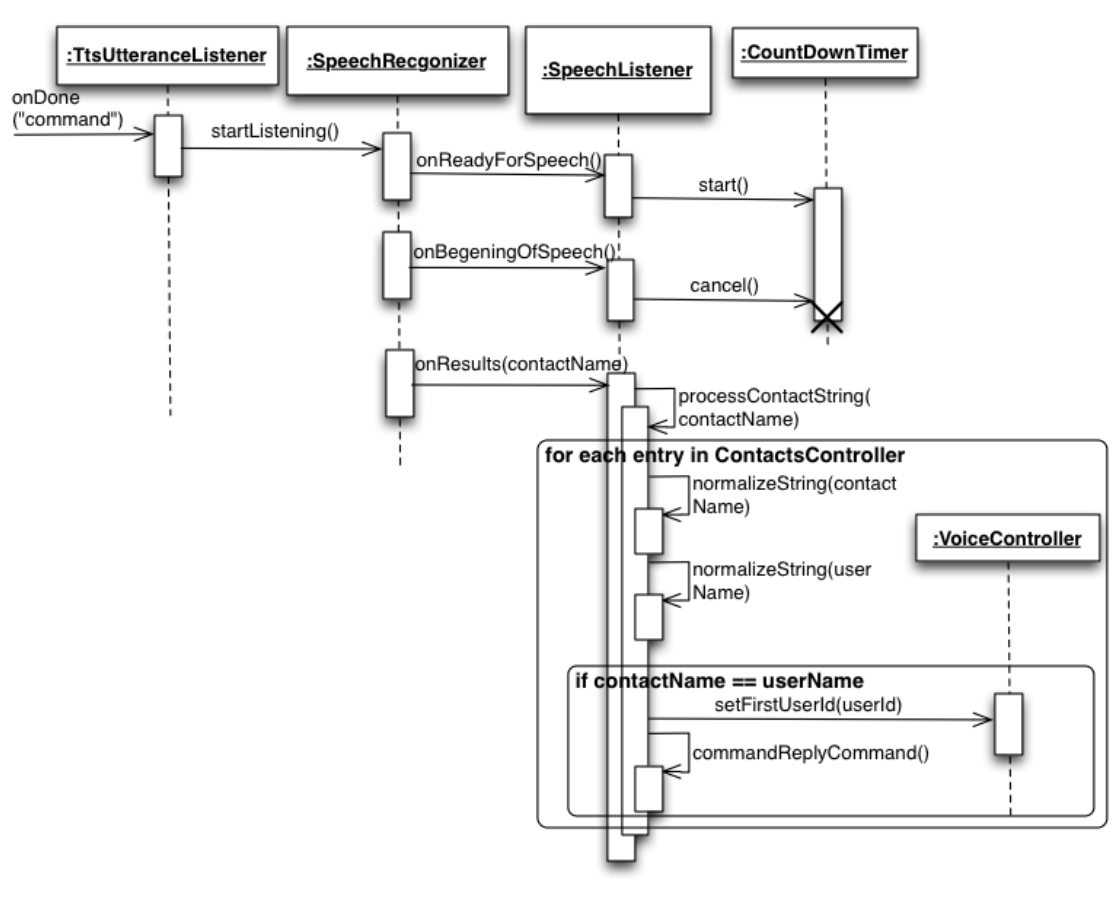

Figura 4.15: Paso 2 del diagrama de secuencia del caso de uso para el comando “Enviar”.

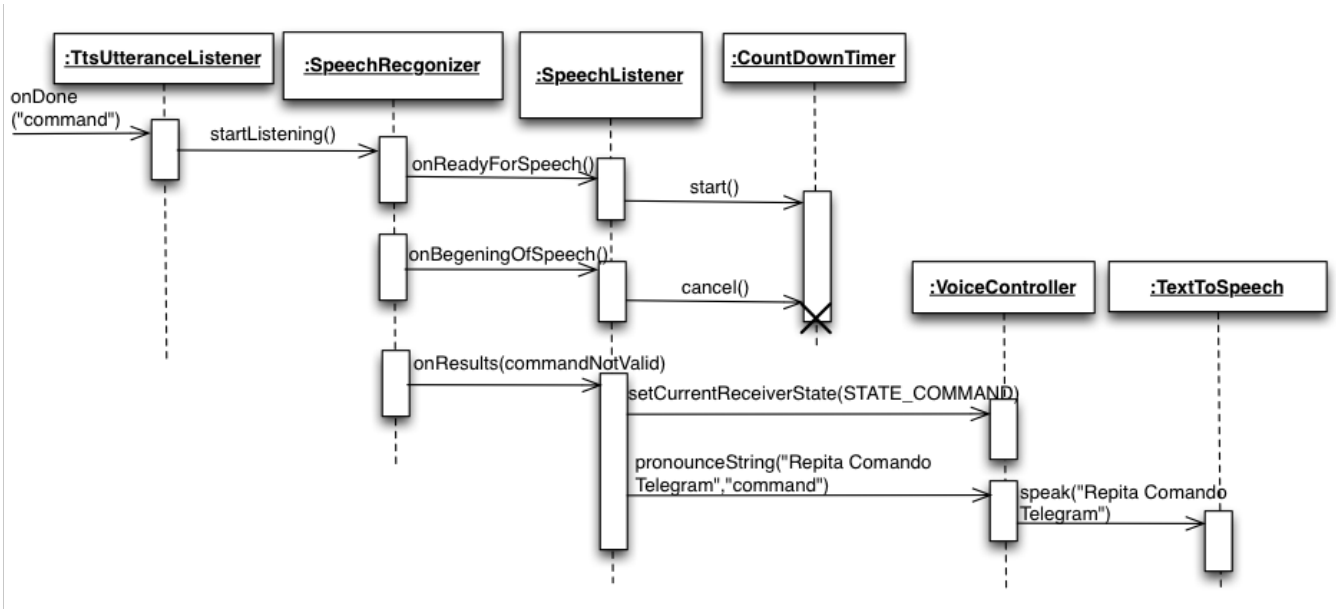

Figura 4.16: Diagrama de secuencia del caso de uso en el que no se reconozca el comando *Telegram*.

La figura 4.16 muestra el caso en el que el usuario no diga un comando correcto y la aplicación no consiga reconocer el comando a realizar. En este caso la aplicación volvería a pedir al usuario decir de nuevo el comando.

Capítulo 5

Pruebas y Resultados

5.1. Metodología de pruebas

El sistema o módulo de Control por Voz ha sido desarrollado de forma iterativa, implementando cada uno de los requisitos funcionales que se definen desde un principio en el análisis de requisitos (3.1). Sin embargo, en nuestro caso muchos requisitos dependían unos de otros y el módulo no podía avanzar a no ser que estuvieran correctamente implementados. Por este motivo, cada vez que se implementaba un requisito funcional se llevaban a cabo pruebas basadas en ese requisito o en los casos de uso que ese requisito implicara. Se verificaba de esta forma si el resultado obtenido se correspondía con lo esperado y definido en la especificación.

En este módulo de Control por Voz , debemos asegurarnos del correcto funcionamiento del *TextToSpeech* y del *SpeechRcognizer*, pues tienen una dependencia y deben llevar a cabo una sincronización. Para probar la integración entre ambas partes será suficiente con probar los casos de uso que se detallarán en el siguiente apartado. Al realizar una prueba por caso de uso nos aseguramos de que cada instancia del código se ejecute al menos una vez, verificando el correcto funcionamiento del módulo completo de Control por Voz integrado en la aplicación cliente de *Telegram*.

Las pruebas se realizaron de forma manual, instalando la versión de *Telegram* modificada por nosotros en un dispositivo móvil que poseía la versión de *Android* 3.2. Previamente a su utilización, se activó la funcionalidad de "Control por voz" en la interfaz de "Ajustes" de la aplicación. Para comprobar que los mensajes eran recibidos correctamente se utilizó otro dispositivo móvil con la versión que existe en el *Play Store* de *Telegram*. A continuación, se detallarán más concretamente los casos de prueba realizados y los correspondientes resultados obtenidos.

5.2. Casos de prueba y Resultados obtenidos

Caso de prueba 1

Descripción

Síntesis de voz de un mensaje recibido en la aplicación de *Telegram*.

La aplicación se encuentra ejecutándose en primer plano y con el chat de la persona de la que se espera recibir el mensaje abierto.

El mensaje deberá quedar escrito en la conversación que corresponda en función del remitente.

Resultado esperado

El mensaje se pronuncia.

1. Se pronuncia el nombre completo del usuario que ha enviado el mensaje.

2. Se pronuncia el mensaje recibido.

3. La aplicación pregunta qué se desea realizar.

El mensaje queda escrito en el chat que se encuentra abierto en ese instante.

Resultado obtenido

Se cumple el resultado esperado.

Requisitos que cumple

R02, R03, R05

Ejemplo de Síntesis de voz

“Mensaje enviado por Lydia Galán: Hola qué tal?”

“Diga un comando Telegram:”

Caso de prueba 2

Descripción

La aplicación no se está ejecutando ni se encuentra en segundo plano. (Aplicación cerrada como muestra la figura 5.1)

Síntesis de voz de un mensaje recibido en la aplicación de *Telegram*.

El mensaje deberá quedar escrito en la conversación que corresponda.

Resultado esperado

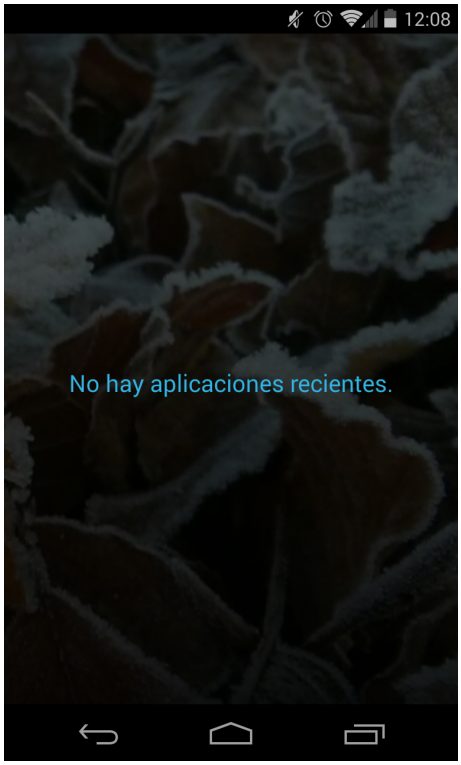

Figura 5.1: Ninguna aplicación ejecutándose en el dispositivo.

El mensaje se pronuncia sin tener la aplicación abierta como muestra la figura 5.2.

1. Se pronuncia el nombre completo del usuario que ha enviado el mensaje.

2. Se pronuncia el mensaje recibido.

3. La aplicación pregunta qué se desea realizar.

El mensaje queda escrito en el chat que se encuentra abierto en ese instante.

Resultado obtenido

Se cumple el resultado esperado.

Requisitos que cumple

R02, R03, R05, R07

Ejemplo de Síntesis de voz

“Mensaje enviado por Lydia Galan: Veamos si funciona con la aplicación cerrada”

“Diga un comando Telegram:”

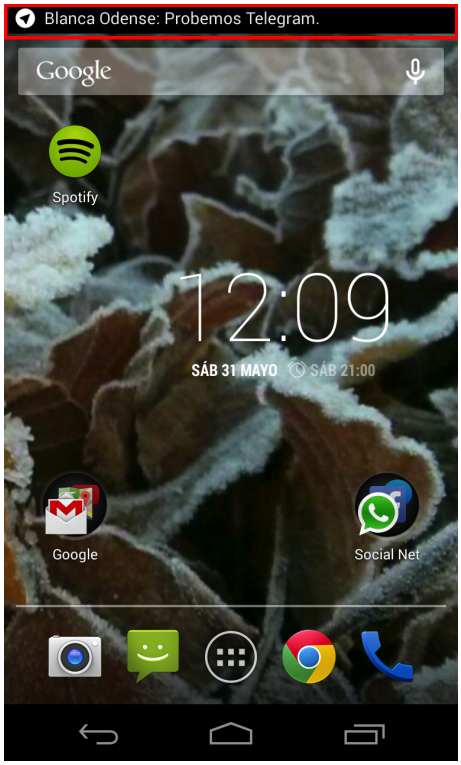

Figura 5.2: Mensaje recibido y pronunciado sin estar ejecutándose *Telegram*.

Caso de prueba 3

Descripción

Lanzamiento y activación del reconocedor de voz una vez se haya producido la síntesis completa del mensaje recibido.

Debe sonar la señal que indica el comienzo del reconocimiento.

Resultado esperado

La señal se activa con éxito. Comienza el reconocedor de voz o *SpeechRecognizer*.

Resultado obtenido

Se cumple el resultado esperado.

Caso de prueba 4

Descripción

El usuario no pronuncia ninguna palabra en un periodo de 5 segundos.

A los 5 segundos comienza a hablar.

No se debe producir ningún reconocimiento de voz.

Resultado esperado

El reconocedor de voz o *SpeechRecognizer* está desactivado. No se produce reconocimiento. El *timer* funciona.

Resultado obtenido

Se cumple el resultado esperado.

Caso de prueba 5

Descripción

Comprobación del correcto funcionamiento del comando "responder" tras la activación del TTS.

Resultado esperado

Usuario pronuncia comando "responder".

TTS pronuncia: "Diga el mensaje"

Usuario pronuncia: "Respondemos al mensaje nuevo"

Mensaje se envía correctamente y aparece en la conversación del contacto que envió un mensaje previo como muestra la figura 5.3.

Resultado obtenido

Se cumple el resultado esperado.

Requisitos que cumple

R02, R03, R05, R07, R08.1

Figura 5.3: Chat del contacto al que hemos respondido el mensaje: “Respondemos al mensaje nuevo”.

Caso de prueba 6

Descripción

Comprobación del correcto funcionamiento del comando “repetir” tras la activación del TTS.

Resultado esperado

Usuario pronuncia comando “repetir”.

TTS pronuncia el último mensaje recibido.

Resultado obtenido

Se cumple el resultado esperado.

Requisitos que cumple

R02, R03, R05, R07, R08.2

Caso de prueba 7

Descripción

Comprobación del correcto funcionamiento del comando “enviar” tras la activación del TTS.

Resultado esperado

Usuario pronuncia comando “enviar”.

TTS pronuncia: “Diga contacto Telegram”

Usuario pronuncia el nombre de un contacto cualquiera de la agenda.

TTS pronuncia: “Diga el mensaje”

El contacto de Telegram recibe el mensaje.

Resultado obtenido

Se cumple el resultado esperado.

Requisitos que cumple

R02, R03, R05, R07, R08.3

Caso de prueba 8

Descripción

Comprobación del correcto funcionamiento de la aplicación al pronunciar cualquier palabra que no sea un comando.

Resultado esperado

Usuario pronuncia comando “Hola”.

TTS pronuncia: “No te he entendido. Repita un comando: responder, repetir o enviar”

Usuario pronuncia “responder”

TTS pronuncia: “Diga el mensaje”

El usuario que envió el último mensaje recibe la contestación.

Resultado obtenido

Se cumple el resultado esperado.

Requisitos que cumple

R02, R03, R04, R05, R07, R08.1

Caso de prueba 9

Descripción

Comprobación del correcto funcionamiento de la aplicación al pronunciar el nombre de un contacto que no existe.

Resultado esperado

Usuario pronuncia comando “enviar”.

TTS pronuncia: “Diga contacto Telegram”

Usuario pronuncia el nombre de un contacto que no existe en su agenda.

TTS pronuncia: “Diga contacto de Telegram”

Usuario pronuncia un contacto de su agenda.

TTS pronuncia: “Diga el mensaje”

El contacto de Telegram recibe el mensaje.

Resultado obtenido

Se cumple el resultado esperado.

Requisitos que cumple

R02, R03, R05, R07, R08.3

Capítulo 6

Conclusiones y Trabajo Futuro

6.1. Conclusiones

A lo largo de este proyecto se han realizado una serie de actividades descritas en esta memoria. Se comenzó analizando cuáles eran los objetivos marcados para el desarrollo del módulo de Control por Voz y se realizó un análisis de los requisitos necesarios para determinar las funcionalidades que debía cumplir dicho módulo. A continuación, fue necesario realizar una familiarización con la plataforma *Android*, dado que nunca antes había trabajado ni desarrollado en esta plataforma. De igual forma, fue necesario hacer un estudio en profundidad de *Telegram*, lo que implicó tener que estudiar muchas líneas de código y funcionalidad para decidir dónde era necesario realizar modificaciones. Por último, se implementó con éxito el módulo de Control por Voz a partir de las librerías de síntesis y reconocimiento de voz que proporciona *Android* y que hasta ese momento también nos eran desconocidas.

Una vez se han conocido mejor los entornos de desarrollo necesarios, se ha llevado a cabo la implementación de las funcionalidades requeridas. Este diseño se ha realizado intentando minimizar el acoplamiento y en su desarrollo se han tenido en cuenta las restricciones presentadas por *Telegram* y las librerías de reconocimiento de voz.

Con el trabajo realizado en este proyecto se ha extendido la funcionalidad de *Telegram* convirtiéndola en la primera aplicación de mensajería que permite una interacción para recibir mensajes y responder a estos, utilizando únicamente la voz. Esta interacción por voz presenta múltiples e importantes beneficios para el usuario: un uso más cómodo del dispositivo móvil, abrir la posibilidad para que personas con discapacidad puedan utilizar esta aplicación, mejorar la seguridad para los usuarios que quieran utilizarla mientras conducen y, en general, podría ser utilizada en cualquier situación en la que la interacción

táctil no sea posible o sea dificultosa.

6.2. Trabajo Futuro

En este último apartado, se describen brevemente una serie de posibles líneas de trabajo futuro. Si bien el módulo implementado podría utilizarse actualmente mientras se está conduciendo ya que ofrece varias posibilidades y comandos, el trabajo desarrollado muestra que en el futuro se podrían implementar más características que complementen las que ya han sido desarrolladas. Algunas de estas nuevas características podrían ser:

- **Soporte para múltiples idiomas**: Nuestro módulo está desarrollado para tres idiomas: Castellano, Inglés y Alemán. Sin embargo, la aplicación es ampliable para los diferentes idiomas que sea necesarios.
- **Activación del módulo con otros eventos**: Actualmente, el módulo de control por voz se activa si el usuario lo hace a través de la interfaz de "Ajustes" de *Telegram*. Esto podría completarse haciendo que otros eventos pudieran ser capaces de activar dicho módulo, como por ejemplo, que la aplicación sea capaz de detectar que el usuario se encuentra en el coche gracias a la velocidad que se presente en ese instante.
- **Pronunciación y detección de emoticonos**: Nuestro módulo es capaz de pronunciar los mensajes recibidos y de responder a estos utilizando únicamente la voz. Sin embargo, si se recibe un emoticono en el mensaje no se tiene constancia de este. Una posibilidad sería detectar el emoticono y traducirlo a voz, y de la misma forma, cuando el usuario pronuncie sentencias como "cara feliz", "cara triste", se traduzcan a sus respectivos emoticonos automáticamente.
- **Añadir más comandos que proporcionen mayor interacción con *Telegram***: Ejemplos que den posibilidad a implementar más comandos podrían ser los siguientes:
  - Al decir el nombre de un contacto de Telegram se pueden producir varias coincidencias. En este caso la aplicación podría dar una lista de los contactos que coinciden para dar opción a elegir a cuál se desea enviar un mensaje.
  - Gestión de ciertas situaciones, como problemas de red en SR.
  - Enviar un mensaje adjuntando una imagen, video o audio grabado.
  - Enviar una ubicación.
  - Enviar información de un contacto.

- Borrar el chat de un contacto determinado.
- Incluir el módulo de Control por Voz para los mensajes encriptados.

Bibliografía

[1] Android Developers. Activities. http://developer.android.com/guide/components/activities.html, 2014.

[2] Android Developers. android.speech - android apis. http://developer.android.com/reference/android/speech/package-summary.html, API added in API level 3.

[3] Android Developers. android.speech.tts - android apis. http://developer.android.com/reference/android/speech/tts/package-summary.html, API added in API level 4.

[4] Benedict Evans. Mobile is eating the world. In *InContext*, 2014.

[5] Centers for Disease Control and Prevention. Distracted driving in the united states and europe. http://www.cdc.gov/Features/dsDistractedDriving/, septiembre 2013.

[6] Centers for Disease Control and Prevention. Mobile device use while driving ? united states and seven european countries, 2011. http://www.cdc.gov/mmwr/preview/mmwrhtml/mm6210a1.htm?s_cid=mm6210a1_w, marzo 2013.

[7] Marko Gargenta and Masumi Nakamura. *Learning Android*. 978-1-449-31923-6. O'REILLY, january 2014.

[8] Android Google Inc. Application fundamentals. http://developer.android.com/guide/components/fundamentals.html, 2014.

[9] J.L.Álvarez. Las distracciones causan el 40 % de los accidentes. http://www.abc.es/sociedad/20130607/rc-distracciones-causan-accidentes-trafico-201306071704.html, junio 2013.

[10] J.M.Sánchez. De whatsapp a telegram: la guerra de las aplicaciones de mensajería instantánea. http://www.abc.

es/tecnologia/moviles-aplicaciones/20140310/abci-whatsapp-telegram-wechat-guerra-201403071850.html, marzo 2014.

[11] Jorge Cordero y otros Manuel Báez, Álvaro Borrego. *Introducción a Android*. 978-84-96285-39-5. E.M.E. Editorial, 2011.

[12] El País. Más de 3.200 conductores denunciados en una semana por usar el móvil. http://politica.elpais.com/politica/2013/06/13/actualidad/1371135961_806806.html, junio 2013.

[13] Flurry Analytics Simon Khalaf. Mobile use grows 115 % in 2013, propelled by messaging. http://blog.flurry.com/bid/103601/Mobile-Use-Grows-115-in-2013-Propelled-by-Messaging-Apps, enero 2014.

[14] Telegram. Articles about telegram. https://telegram.org/press, 2013.

[15] Telegram. Mtproto mobile protocol. https://core.telegram.org/mtproto, 2013.

[16] Telegram. Telegram - taking back our right to privacy. https://telegram.org/, 2013.

[17] Telegram. Telegram messenger for android source, github. https://github.com/DrKLO/Telegram, 2014.

[18] Wikipedia. Line (application) — wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Line_(application)&oldid=605270086, 2014. [Online; accessed 24-April-2014].

[19] Wikipedia. Telegram (software) — wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Telegram_(software)&oldid=603318305, 2014. [Online; accessed 24-April-2014].

[20] Wikipedia. Whatsapp — wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=WhatsApp&oldid=605553365, 2014. [Online; accessed 24-April-2014].

[21] Martin Fowler with Kendall Scott. *UML Distilled*. ISBN 0-201-65783-X. Addison Wesley Longman, Inc, 2000.

[22] G.Blake Meike Zigurd Mednieks, Lara Dornin and Masumi Nakamura. *Programming Android*. 978-1-449-38969-7. O'REILLY, july 2011.

Este documento esta firmado por

	**Firmante**	CN=tfgm.fi.upm.es, OU=CCFI, O=Facultad de Informatica - UPM, C=ES
	Fecha/Hora	Fri Jun 06 19:00:26 CEST 2014
	Emisor del Certificado	EMAILADDRESS=camanager@fi.upm.es, CN=CA Facultad de Informatica, O=Facultad de Informatica - UPM, C=ES
	Numero de Serie	630
	Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)